Complex Trigger Unit for Nexus MPC5xxx

Complex Trigger Unit for Nexus MPC5xxx

Version 18-Nov-2020

Introduction

This application note describes the features and programming of the Complex Trigger Unit for MPC5XXX processors with a parallel Nexus trace port.

The PowerTrace module contains the Complex Trigger Unit, short CTU. The CTU is a trigger sequencer that provides additional trigger and filter possibilities. The usage of the CTU has no impact on the real-time behavior of the processor.

The Complex Trigger Unit is only implemented for parallel NEXUS interfaces with trace port widths **MDO8 | MDO12 | MDO16.**

The CTU does not support on-chip traces or Aurora NEXUS trace ports.

The CTU is programmed by a special trigger language.

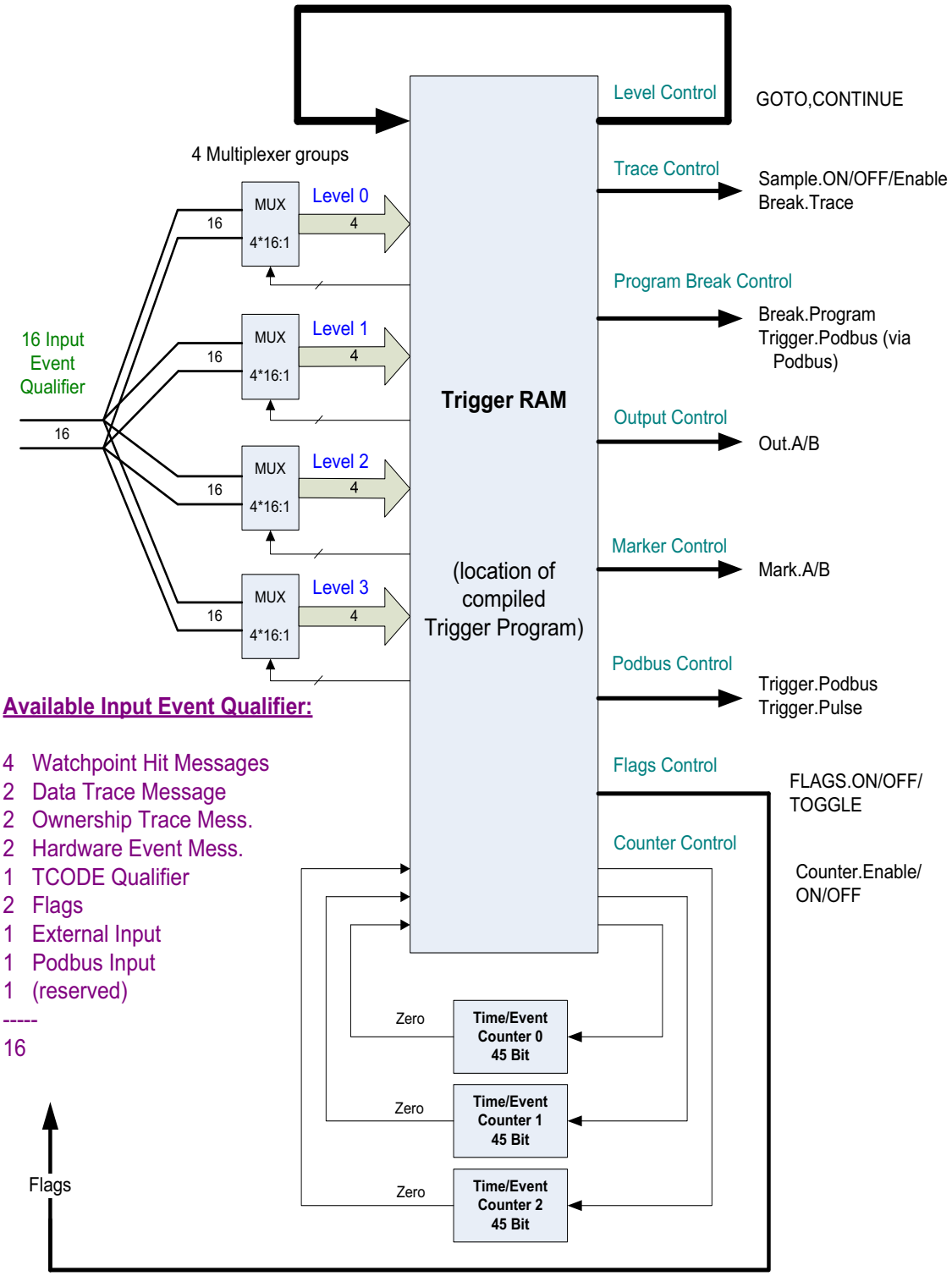
The main input events for the CTU together with a NEXUS adapter are:

- Watchpoint hit messages
- Data trace messages
- Ownership trace messages
- External trigger input pin

Based in this input events, the CTU can provide the following features:

- Trigger output signals
- Halt program execution
- Halt trace recording (trace trigger)
- Trace filtering
- Set marks in trace recording

Complex Trigger Unit (CTU) Diagramm



Available Input Event Qualifier:

4 Watchpoint Hit Messages
2 Data Trace Message
2 Ownership Trace Mess.
2 Hardware Event Mess.
1 TCODE Qualifier
2 Flags
1 External Input
1 Podbus Input
1 (reserved)

16

Data Trace Message based events

Stop the sampling to the trace buffer if a 1 as a byte is written to the variable flags[3].

To declare the input event - write of byte 1 to the address flags[3] - a **Data Trace Message Qualifier** has to be used. The CTU provides 2 Data Trace Message Qualifiers. The 2 Data Trace Message Qualifiers can be used to qualify 2 single Data Trace Message Qualifiers or to qualify 1 Data Trace Message Qualifier for an address range.

The special feature of a Data Trace Message Qualifier is that the full data address and the full data value is reconstructed by the PowerTrace hardware in real-time out of the compressed information provided by the Data Trace Messages.

Data Trace Message Qualifiers have to be declared before they can be used in a trigger program.

Format: **ADDRESS** <*selector*> <*address*>| <*range*> ***/HARD*** [<*option*> …]

<*selector*>: **AlphaBreak**
BetaBreak
CharlyBreak
DeltaBreak
EchoBreak

<*option*>: **Read | Write | ReadWrite**
Data.auto <*value*> **| Data.Byte** <*value*> **| Data.Word** <*value*> **|**
Data.Long <*value*>

```
ADDRESS AlphaBreak 0x7403             /HARD /Write /DATA.Byte 0x01

ADDRESS AlphaBreak V.RANGE(flags[3]) /HARD /Write /DATA.Byte 0

ADDRESS AlphaBreak V.RANGE(flags)    /HARD /Write /DATA.Byte 1
```

The CTU can either stop the sampling to the trace buffer or the program execution when the trigger event defined by the Data Trace Message Qualifier occurs.

Format: **BREAK.TRACE | BREAK.PROGRAM** [**IF** <*condition*>]

Example: Trace trigger on data value

Full example for stopping the sampling to the trace buffer:

```
NEXUS.DTM Write          ;enable data trace messaging for write accesses

Analyzer.ReProgram       ;set trigger program
(
  ADDRESS AlphaBreak V.RANGE(flags[3]) /HARD /WRITE /DATA.Byte 1
  BREAK.TRACE IF AlphaBreak
)

Go                       ;run application
```

Example: Program break on data value

Full example for a data value breakpoint on MPC55XX (e200z1, z3, z6 cores do not provide DVC). In order to prevent FIFO overflows, data trace is limited to the variable of interest. The stop of the program execution is delayed (asynchronous stop). For this reason the defined event is marked with an “A” marker in the trace listing.

```
Var.Break.Set flags[3] /Write /TraceData ;data trace only for flags[3]
TrOnchip.EVTI ON         ;use EVTI to halt the core as fast as possible

Analyzer.ReProgram       ;set trigger program
(
  ADDRESS AlphaBreak V.RANGE(flags[3]) /HARD /WRITE /DATA.Byte 1
  BREAK.Program IF AlphaBreak
  MARK.A        IF AlphaBreak
)

Go                       ;run application

WAIT !STATE.RUN()        ;wait until core halted
Analyzer.List MARK DEFault ;show program flow with markers
```

The reconstruction of the full data address and the full data value by the PowerTrace hardware can fail if too many data trace messages are generated in quick succession.

Watchpoint hit message based events

Analyze the run-time behavior of the function sieve.

The basic ideas for this analysis are:

- generate a watchpoint hit message (WHM) at the entry and at the exit of the function sieve.
- sample only these watchpoint hit messages to the trace buffer.
- to differentiate between the watchpoint hit messages generated for the function entry and those generated for the function exit, mark the Watchpoint Trace Messages generated for the function entry with an A marker and the Watchpoint Trace Messages generated for the function exit with a B marker

Watchpoint hit messages have to be declared before they can be used in a trigger program.

Format:	**ADDRESS** <*selector*> <*address*>| <*range*> **/Onchip /Program**
<*selector*>	**AlphaBreak** **BetaBreak** **CharlyBreak** **DeltaBreak** **EchoBreak**

```
ADDRESS AlphaBreak sieve              /Program /Onchip

ADDRESS BetaBreak sYmbol.EXIT(sieve) /Program /Onchip
```

The following trigger instructions are available to control the sampling to the trace buffer:

Format:	**Sample**[.<*mode*>] [**IF** <*condition*>]
<*mode*>:	**Enable** **OFF** **ON**

Enable Releases trace memory for recording when the specified condition is true.

OFF Switch the sampling to the trace buffer to OFF

ON Switch the sampling to the trace buffer to ON.

The following trigger instructions are available to mark a record in the trace buffer:

Format:	**Mark**[**.**<*marker*>] [**IF** <*condition*>]
<*marker*>:	**A** **B**

Example: Runtime measurement with markers

Now the full example:

```
NEXUS.BTM OFF           ;optional: disable program/data trace for maximum
NEXUS.DTM OFF           ;accuracy (only watchpoint messages are used)

Analyzer.ReProgram      ;set trigger program
(
  ADDRESS AlphaBreak sYmbol.BEGIN(sieve) /Program /Onchip
  ADDRESS BetaBreak  sYmbol.EXIT(sieve)  /Program /Onchip

  Mark.A IF AlphaBreak
  Mark.B IF BetaBreak
)

Go                                ;run application
WAIT 2.s
Break

Analyzer.STATistic.DURation       ;display a function run-time statistic
Analyzer.PROfileChart.DURation    ;display a function run-time chart
```

If the run-time analysis of the function sieve showed, that the function sieve took several times much longer then you expected, a new trigger program can help you to find the reason for this behavior.

The basis idea of this new trigger program is:

- stop the program execution if the function sieve takes longer the 80 µs.

To write this trigger program 2 new concepts of the trigger programming language are required:

- Time Counters
- Trigger Levels

Time Counters: The CTU provides 3 45-bit time counter with a resolution of 20 ns. Before a Time Counter can be used in a trigger program it has to be declared.

Format: **TImeCouNTer** <name> [<time>]

```
TImeCouNTer sievec              80.us

TImeCouNTer interrupt_response 10.ms
```

The following trigger instructions can be used to control the Time Counters:

Format: **Counter**[.<*mode*>] <*counter_name*> [**IF** <*condition*>]

<*mode*>: **Increment**
OFF
ON
Restart

Increment Increment the counter if the specified condition is matched.

OFF Switch the counter to ON if the specified condition is matched.

ON Switch the counter to OFF if the specified condition is matched.

Restart The counter is reset to zero if the specified condition is matched.

If a Time Counter is used in a condition, the Time Counter is a true event when it has reached the declared time value. The current contents of a Time Counter can be see in the **Trace.state** window.

```
Counter.Increment sievec             IF AlphaBreak

Counter.Restart   interrupt_response IF BetaBreak

Break.Program                        IF sievec
```

Trigger Level: The CTU provides 4 trigger levels.

- A trigger level starts at its label.
- A trigger level ends at the following label or at the end of the trigger program.
- The levels determine which trigger instructions are active at the same time.

Changing a trigger level is done by the following trigger instruction:

Format:	**GOTO** <*level*> [**IF** <*condition*>]

Example: Program break based function runtime

Here the full example:

```
Analyzer.ReProgram
(
  ADDRESS AlphaBreak sYmbol.BEGIN(sieve) /Program /Onchip
  ADDRESS BetaBreak  sYmbol.EXIT(sieve)  /Program /Onchip

  TImeCouNTer sievec 80.us

  start:
    GOTO insieve IF AlphaBreak

  insieve:
    Counter.Increment sievec
    Counter.Restart   sievec, GOTO start IF BetaBreak
    BREAK.PROGRAM IF sievec&&!BetaBreak
)

Go
WAIT !STATE.RUN()
PRINT "Function sieve exceeded maximum runtime!"
```

The complex trigger unit also provides flags to store an internal state. This state can be used as element of conditions.

Format:	**Flag**[.<*action*>] [**IF** <*condition*>]
<*mode*>:	**FALSE** **TRUE** **Toggle**

Using external signals with the CTU

The CTU supports two input signals. The IN input of the NEXUS adapter, and the PodBus trigger signal.

The IN input is labeled “IX0” on the NEXUS AutoFocus adapter LA-7630 and “IN0” on LA-7610.

The PodBus trigger signal can be either an internal source (Debug module, Power Probe or Power Integrator logic analyzers) or it can stem from an external source through the “Trigger” connector of the debug module.

Format:	[<*action*>] [**IF** <*condition*>]	
<*condition*>:	**BUSA | !BUSA**	(PodBus trigger signal)
	IN | !IN	(IN connector of NEXUS adapter)

There are also output signals available. The OUT output of the NEXUS adapter, and the PodBus trigger signal.

The OUT output is labeled “OX0” on the NEXUS AutoFocus adapter LA-7630 and “OUT0” on LA-7610.

The PodBus trigger signal can trigger any device on the PodBus (Debug module, Power Probe or Power Integrator logic analyzers). It can also be used to trigger external devices using the “Trigger” connector of the debug module.

Format:	[<*action*>] [**IF** <*condition*>]	
<*action*>:	**TRIGGER.PODBUS**	(PodBus trigger signal)
	OUT.A	(OUT connector of NEXUS adapter)

Example: Record single message on rising edge of trigger input

The next example demonstrates how to control trace recording based on an external signal connected to the Trigger input of the debug module.

```
Analyzer.ReProgram
(
waitrisingedge:
  Sample.Enable        IF BUSA
  GOTO waitfallingedge IF BUSA

waitfallingedge:
  GOTO waitrisingedge  IF !BUSA
  )
)

;configure Trigger connector as high-active input
TrBus.Connect In
TrBus.Mode HIGH
```

Example: Program break based on pulse interval of IN input

The next example demonstrates how to use a flag to monitor a state change. Mark B is set on every rising edge of the IN signal. The program execution is halted if the time interval of two rising edges is shorter than 10ms.

```
Analyzer.ReProgram
(
  TImeCouNTer interval 10.ms
  FLAGS       lastin

  ;FLAG lastin is value of IN, delayed by one CTU cycle
  FLAG.TRUE   lastin       if IN
  FLAG.FALSE  lastin       if !IN
  ;generate MARK.B on rising edge of IN
  MARK.B                   if IN&&!lastin

waitnext:
  Counter.ON      interval
  Counter.Restart interval if IN&&!lastin
  GOTO criticaltime        if IN&&!lastin

criticaltime:
  GOTO waitnext if interval               ;interval elapsed -> OK
  GOTO timeviol if IN&&!lastin&&!interval ;pulse within interval -> fail

timeviol:
  Counter.OFF interval
  BREAK.PROGRAM
)

Go
WAIT !STATE.RUN()
PRINT "Found two pulses within one 10ms interval"
Analyzer.List Trigger.0 MARK.B TIME.MARKBB DEFault
```

Input Event	Meaning
IN	external input event IN0 or IN1 occurred
CM, TCODE_2, TCODE_CM	correlation message
DBM, TCODE_3, TCODE_DBM	direct branch message
DBSM, TCODE_B, TCODE_DBSM	direct branch sync message
DRM, TCODE_6, TCODE_DRM	data read message
DRSM, TCODE_E, TCODE_DRSM	data read sync message
DSM, TCODE_0, TCODE_DSM	debug status message
DWM, TCODE_5, TCODE_DWM	data write message
DWSM, TCODE_D, TCODE_DWSM	data write sync message
EM, TCODE_8, TCODE_EM	error message
EM_0, TCODE_8_0	error message 0 - OTM loss
EM_1, TCODE_8_1	error message 1 - BTM loss
EM_2, TCODE_8_2	error message 2 - DTM loss
EM_3, TCODE_8_3	error message 3 - r/w access error
EM_5, TCODE_8_5	error message 2 - invalid access opcode
EM_6, TCODE_8_6	error message 6 - WHM loss
EM_7, TCODE_8_7	error message 7 - BTM/DTM/OTM loss
EM_8, TCODE_8_8	error message 8 - BTM/DTM/OTM/WHM loss
EM_24, TCODE_8_24	error message 24
EM_31, TCODE_8_31	error message 31
HBM, TCODE_1C, TCODE_HBM	hardware event message
HWM, TCODE_38, TCODE_HWM	hardware event message
HWSM, TCODE_1D, TCODE_HWSM	hardware event sync message
IBM, TCODE_4, TCODE_IBM	indirect branch message
IBSM, TCODE_C, TCODE_IBSM	indirect branch sync message
OTM, TCODE_2, TCODE_OTM	ownership trace message

RBM, TCODE_1E, TCODE_RBM	repeat branch message
RFM, TCODE_1B, TCODE_RFM	resource full message
WHM, TCODE_F, TCODE_WHM	watchpoint hit message

For not CPU-specific keywords, see **non-declarable input variables** in **"ICE/FIRE Analyzer Trigger Unit Programming Guide"** (analyzer_prog.pdf).