Freescale Semiconductor
Application Note

AN2857
Rev. 0, 11/2004

Using the Queued Output Match (QOM) eTPU Function

by: Geoff Emerson
Freescale 32-Bit Embedded Controller Division

This eTPU Queued Output Match (QOM) Application Note is intended to provide simple C interface routines to the QOM eTPU function. The functions are usable on any product which has an eTPU module. Example code is available for the MPC5554 and MCF5235 devices. This application note should be read in conjunction with application note AN2864, “General C Functions for the eTPU.”

Table of Contents

1 Function Overview

The QOM function generates complex output pulse trains without CPU intervention using a sequence of output matches. An output match occurs when a user-defined value is matched by the value of an internal timebase. When an output match occurs, a user-specified pin state is driven on the output pin. The eTPU QOM function generates multiple output matches using a table of offset times. These offset times, along with the corresponding pin states, are stored in Data Memory. The table size is user-programmable. Various modes of queue operation are supported.

The eTPU QOM function is based on the QOM TPU function. The QOM eTPU function offers the following enhancements over the QOM TPU function:

- Table size is limited only by the amount of Data Memory available on the target product
- 22-bit offset values are supported (versus 15-bit offset value on the TPU3)

2 Functional Description

Entries in the QOM queue (event table) are relative match offsets, not absolute match times. The next match time in a sequence is calculated by adding the next offset in the table to the time of the last match. If the match is the first match in a sequence, the first offset value in the table is added to a selectable reference time.

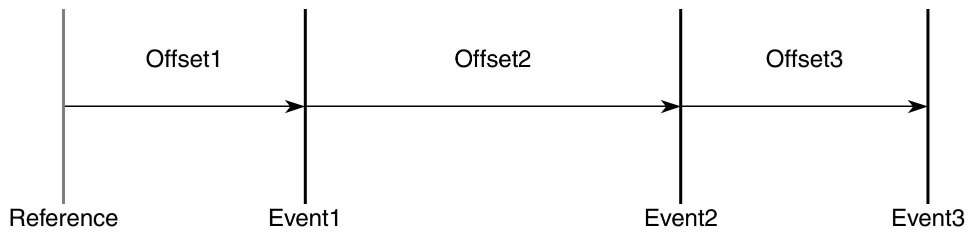


Figure 1. References, Events and Offsets

The reference from which the first match in a sequence is scheduled can be the immediate value of the selected timebase (Timer Counter Register 1 or 2: TCR1/2), the time of the last match of a previous sequence, or a reference contained in Data Memory. Using the time of the last match of a previous sequence as a reference allows a series of sequences to be chained together. Using a reference from Data Memory allows a sequence of output matches to be referenced to a value supplied by another eTPU channel.

Pin state (high or low) when a match occurs is programmable. Pin state is determined by the value of the LSB in each table entry.

The function can operate in three modes: single shot mode, loop mode, and continuous mode. In single-shot operation mode, a sequence of match outputs is generated once. In loop operation mode, a sequence of match outputs is generated a specified number of times (1 to 256). In continuous operation mode, the entire sequence repeats until the channel is disabled. All these modes can be used in conjunction with linked operation. Linked operation allows the function to be triggered by a link from another eTPU channel.

In single-shot and loop operation modes, the event time of the last match in the table is written back into Data Memory, which can be accessed by the CPU. During initialization, the pin can be configured to be high, low, or no change. Matches are scheduled using both of the eTPU's action units. Each match offset can have a maximum value of 0x40 0000. This allows the second future match to be up to 0x80 0000 in the future.

If more than two table offset values are programmed for the same pin state, the duration of an output event can effectively be extended beyond the normal 0x80 0000 count limit.

2.1 Notes on the Performance and Use of eTPU QOM Function

2.1.1 Performance

Like all eTPU functions, QOM function performance in an application is to some extent dependent upon the service time (latency) of other active eTPU channels. This is due to the operational nature of the eTPU scheduler. When a single QOM channel is in use and no other eTPU channels are active, the minimum time between sustained successive matches is less than 38 eTPU clock cycles in single shot and loop operation modes, and less than 29 eTPU clocks in continuous operation mode. (These figures include 3 eTPU clocks for Time Slot Transition.). It is, however, possible to program the hardware so that two consecutive events occur 1 eTPU clock apart. Subsequent edges programmed to occur less than the latency amounts will actually occur later than programmed. When more eTPU channels are active, performance decreases. However, worst-case latency in any eTPU application can be closely estimated. To analyze the performance of an application that appears to approach the limits of the eTPU, use the guidelines given in the eTPU reference manual and the information provided in the eTPU QOM software release available from Freescale.

2.1.2 Using QOM for Pulse Width Modulation

The eTPU QOM function can also be used to generate a pulse-width modulated output in systems that do not have a dedicated eTPU PWM (Pulse Width Modulation) function. A PWM output is generated using continuous operation mode and two match offsets. One offset is configured to generate a rising edge, and the other is configured to generate a falling edge. The offset that generates the rising edge is the low-time parameter, and the offset that generates the falling edge is the high-time parameter. Modulation is achieved by varying the offset values in the table. 100% and 0% duty cycles are easily obtained by configuring both offsets to generate either a rising edge (100% duty cycle) or a falling edge (0%).

2.1.3 Effect of Using a Reference which is in the Past

When a time contained in Data Memory is used as a reference for the first match event, it is possible that this time is in the past. Under these circumstances, the behavior of the function is dependant upon how far in the past the reference is relative to when the eTPU services the host service request. Figures 2 through 5 show the behavior of the function for various reference values. When the eTPU engine services the host service request two or more events 'late,' then the first two events will be skipped. Note that 'past' and 'future' are relative to the time of the Host Service Request (HSR).

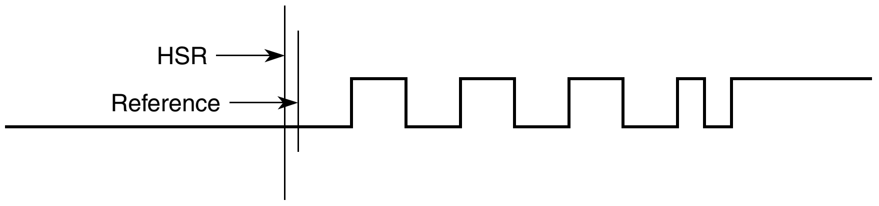


Figure 2. The Reference in the Future; All Programmed Events Occur

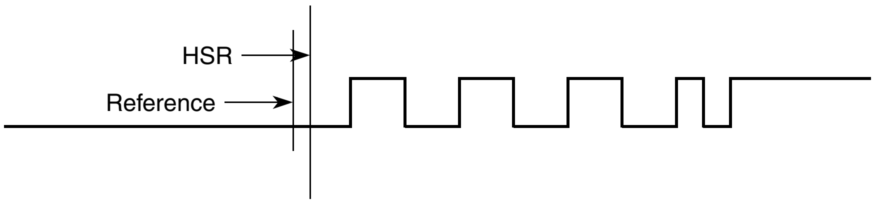


Figure 3. The Reference in the Past and the First Two Events in the Future; All Programmed Events Occur

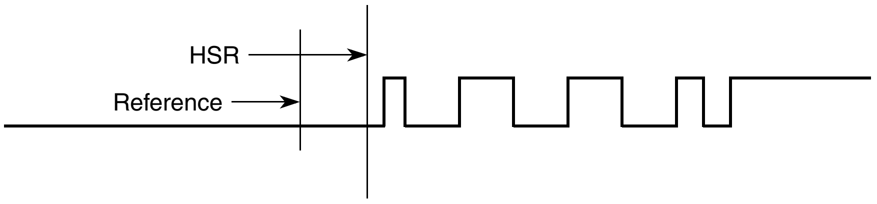


Figure 4. Here the reference is in the past. The first event is in the past while the second event is programmed to be in the future. All programmed events occur, although all events are delayed relative to reference. In this case, the first positive pulse is shortened because reference+offset1 is in the past and causes an immediate match. Reference+offset1+offset2 is in the future and events proceed normally.

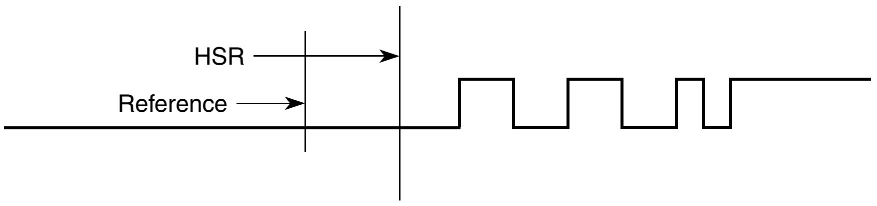


Figure 5. Here the reference is in the past. The first 2 events are in the past. First two events are effectively skipped. Since (reference+offset1) and (reference+offset1+offset2) both occurred in the past, they cause immediate matches. However, because their pin actions are opposing (and therefore cancel), both events appear not to have happened.

2.1.4 Effect of Latency on Short Table Entries

If two or more consecutive event table entries are shorter than the service times detailed in the information provided in the eTPU QOM software release available from Freescale, then the events after the first short event will be delayed. All subsequent events will also be delayed. The diagrams below show the ideal versus actual behavior if the event table were as follows:

TCR counts	Pin State
100	HIGH
100	LOW
1	HIGH
1	LOW
100	HIGH
100	LOW

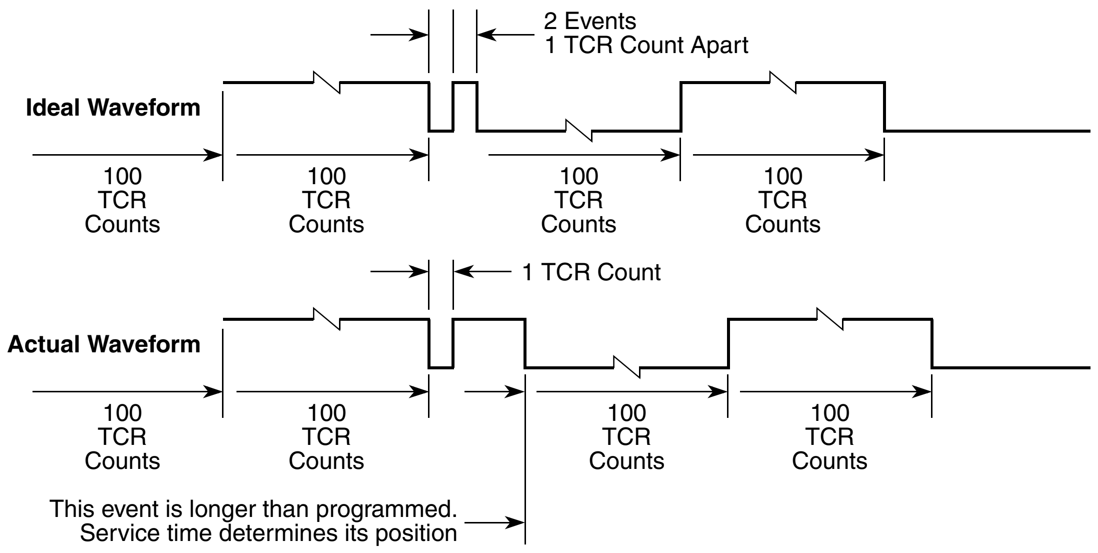


Figure 6. Real v. Ideal Waveforms

2.1.5 Linked Operation

A channel may be initialized to receive a link from another eTPU channel. When a link is received, the QOM function will function as if a host service request had been received.

During linked operation, if an additional link is received after the function has started running, the link thread is re-executed. Executing the link thread again causes the match sequence to be started at the beginning of the table. When this happens, a new match replaces any pending match. If an additional link is received while a loop is executing, the match sequence is restarted at the beginning of the table. The total number of whole loops executed to completion does not change as a result of the second link being received. When a complete match sequence has been executed and the function has stopped, additional link requests are ignored.

2.1.6 Changing Operation Modes

In order to change operation modes on the channel while it is still running, the channel must first be disabled. This can be done using the fs_etpu_disable function, found in file etpu_utils.h.

2.1.7 Using a Reference Address for the First Match

The examples in this document that use a reference time pointed to by *ref have reference times that occur before the HSR or a link that initiates the function. This need not be the case. A first match is scheduled correctly when the HSR or link occurs before the reference time, as shown in Figure 7.

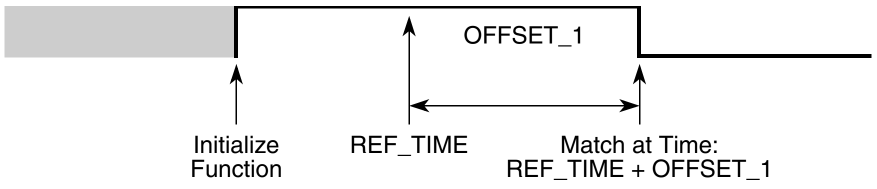


Figure 7. First Match with HSR Prior to Reference Time

It should be noted that in order to avoid improper operation the following condition must be met:

Reference + first offset amount + second offset amount < 0x80 0000 + current TCR value

If this condition is not met, the first events will be scheduled in the past and cause immediate matches to occur.

2.1.8 Long Match Times

The eTPU QOM function can produce long apparent match times by using two or more table entries programmed for the same pin response. This technique can extend a pin state change well beyond the normal limit of 0x3F FFFF TCR counts. Because very short match times can have an adverse effect on overall eTPU performance, it is best to split a long match time into even segments rather than following a very long segment with a very short segment. For example, if a total match time of 0x40 0100 counts is required, use two 0x20 0080-count match times rather than one match time of 0x3F FFFF counts and another of 0x101 counts.

3 C Level API for eTPU QOM Function

The following routines provide easy access for the application developer to the QOM function. Use of these routines eliminates the need to directly control the eTPU registers. The API consists of one function. This function can be found in the etpu_qom.h and etpu_qom.c files. The routines are described below and are available from Freescale. In addition, the eTPU compiler generates a file called etpu_qom_auto.h. This file contains information relating to the eTPU QOM function including details of how the Data Memory is organized and definitions for various API parameters.

- Initialization Routine:

```
void fs_etpu_qom_init        (unit8_t channel,
                              uint8_t priority,
                              uint8_t mode,
                              uint8_t timebase,
                              uint8_t init_pin,
                              uint8_t first_match_mode,
                              uint32_t *ref,
                              uint8_t Loop,
                              uint8_t event_array_size,
                              union etpu_events_array event_array[]);
```

3.1 Initialization Routine

Initialization routine fs_etpu_qom_init dynamically allocates Data Memory. If dynamic allocation is not required, then the clock channel's Channel Parameter Base Address field should be written with a non-zero value before calling the fs_etpu_spi_init function.

Dynamic allocation of Data Memory occurs if the channel has a zero in its Channel Parameter Base Address field. The Channel Parameter Base Address field is updated by the API with a non-zero value to point to the parameter RAM allocated to the channel. The fs_etpu_qom_init API will not allocate new parameter RAM if the channel has a non-zero value in its Channel Parameter Base Address field; this means that channel has already been assigned. If the user needs to reuse the channel with a different event table, care must be taken to ensure that the event_array_size parameter is not greater than the original.

This routine is used to initialize an eTPU channel for the eTPU QOM function. After the channel has been initialized the QOM queue will be executed as specified. If the channel has been initialized to receive a link, the queue will not be executed until a link has been received. This function has the following parameters:

- **channel (uint8_t)**: The QOM channel number. For devices with two eTPUs, this parameter should be assigned a value of 0-31 for eTPU_A and 64-95 for eTPU_B. For products with a single eTPU, this parameter should be assigned a value of 0-31.
- **priority (uint8_t)**: The priority to assign to the eTPU QOM channel. The following eTPU priority definitions are found in utilities file etpu_utils.h.**:**
  - FS_ETPU_PRIORITY_HIGH
  - FS_ETPU_PRIORITY_MIDDLE
  - FS_ETPU_PRIORITY_LOW
  - FS_ETPU_PRIORITY_DISABLED

- **mode (uint8_t):** The operation mode in which the eTPU QOM function will run. The following eTPU QOM operation mode definitions are found in the etpu_qom_auto.h file:
  — FS_ETPU_QOM_SINGLE_SHOT
  — FS_ETPU_QOM_LOOP
  — FS_ETPU_QOM_CONTINUOUS
  — FS_ETPU_QOM_CONTINUOUS_A
- **timebase (uint8_t):** The timebase the eTPU QOM channel will use. The following eTPU timebase definitions are found in utilities file etpu_utils.h:
  — FS_ETPU_TCR1
  — FS_ETPU_TCR2
- **init_pin (uint8_t):** The pin state at function initialization. The following eTPU QOM initial pin state definitions are found in the etpu_qom_auto.h file:
  — FS_ETPU_QOM_INIT_PIN_LOW
  — FS_ETPU_QOM_INIT_PIN_HIGH
  — FS_ETPU_QOM_INIT_PIN_NO_CHANGE
  — FS_ETPU_QOM_INIT_PIN_LOW_LINK
  — FS_ETPU_QOM_INIT_PIN_HIGH_LINK
  — FS_ETPU_QOM_INIT_PIN_NO_CHANGE_LINK
  The definitions which have “LINK” in their names are used for link mode operation.
- **first_match_mode (uint8_t):** The reference for the first match. The following eTPU QOM first match mode definitions are found in the etpu_qom_auto.h file:
  — FS_ETPU_QOM_IMMEDIATE
  — FS_ETPU_QOM_USE_LAST_EVENT
  — FS_ETPU_QOM_USE_REF_ADDRESS
- **ref (uint32_t*):** The address of the Data Memory location whose contents will be used as a reference when first_match_mode is FS_ETPU_QOM_USE_REF_ADDRESS.
- **loop (uint8_t):** The number of times the event table should be repeated when the channel is running in loop operation mode (mode = FS_ETPU_QOM_LOOP). If loop =0 then 256 iterations are performed.
- **event_array_size (uint8_t):** The number of entries in the events table.
- **event_array []:** The table of events used to define the output signal.

If this initialization routine is called more than once, the user needs to ensure that the table length is not greater then when the channel was first initialized. Otherwise the parameters for the channel stored in Data Memory above the QOM channel Data Memory will be corrupted.

4 Examples of Function Use

4.1 Simple Example

4.1.1 Description

This section describes a simple use of the QOM function and how to initialize the eTPU module and assign the eTPU QOM function to an eTPU channel.

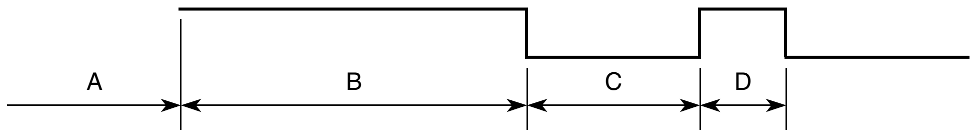
4.1.2 Example Code

The example consists of two files:

- qom_example1.h
- qom_example1.c

File qom_example1.c contains the main() routine. This routine initializes the MPC5554 device for 128 MHz CPU operation and initializes the eTPU according to the information in the my_etpu_config struct (stored in file qom_example1.h). The pins used in this example are configured for eTPU operation, then the QOM function is initialized on channel QOM0 (ETUA0). The channel is configured with medium priority in loop operation mode using TCR1. The state of the pin after initialization but prior to the first event will be logic low. The events will be scheduled immediately. Since reference address mode for the first event is not being used, a dummy pointer is passed. The Loop parameter is set to 3.

Basic waveform used in this example (as defined by my_QOM_event_arrayA; used to build the event table used by the QOM function).



A	Pin High After 0x010000 TCR1 Counts
B	Pin Low After 0x008000 TCR1 Counts
C	Pin High After 0x004000 TCR1 Counts
D	Pin Low After 0x002000 TCR1 Counts

4.1.3 Program Output

Figure 8 shows the output on QOM0 (ETPUA0) captured from a logic analyzer which the sample program generated.

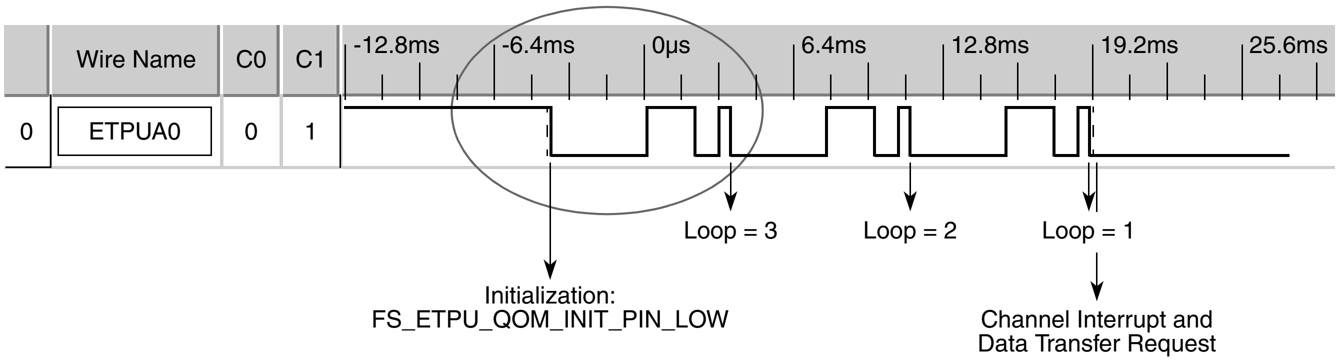
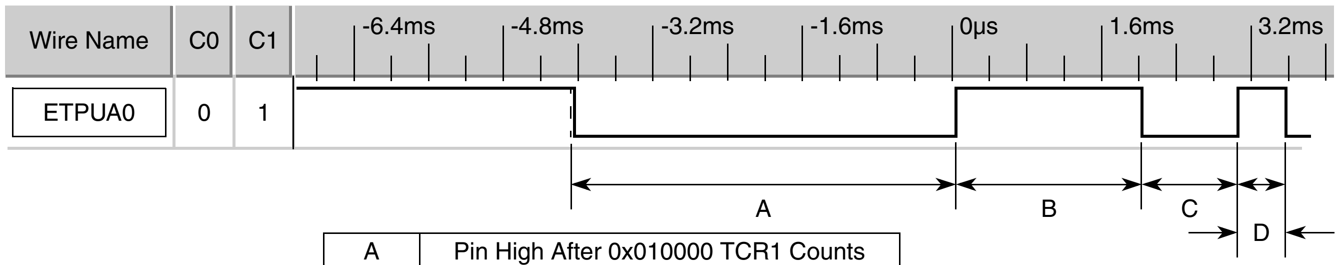


Figure 8. qom_example1.c QOM0 (ETPUA0) Output



A	Pin High After 0x010000 TCR1 Counts
B	Pin Low After 0x008000 TCR1 Counts
C	Pin High After 0x004000 TCR1 Counts
D	Pin Low After 0x002000 TCR1 Counts

Figure 9. Zoom-In on Area Encircled in Figure 8

4.2 More Complex Example

4.2.1 Description

This section describes a more complex use of the QOM function. Six channels are initialized to run the QOM function. The channels are configured to demonstrate various different features of the function. In this example, use is made of the eTPU TEST function to generate the links required to demonstrate link operation.

4.2.2 Example Code

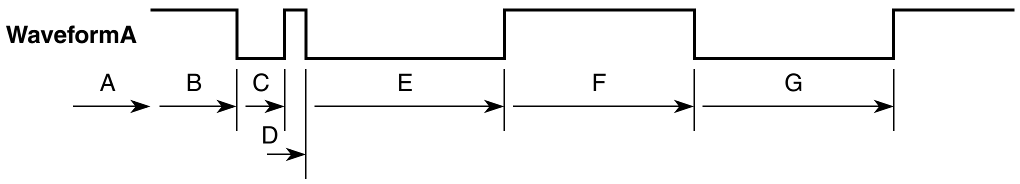
The example consists of two files:

- qom_example2.h
- qom_example2.c

File qom_example2.c contains the main() routine. This routine initializes the MPC5554 device for 128 MHz CPU operation, then initializes the eTPU according to the information in the my_etpu_config struct (stored in file qom_example2.h). The pins used in this example are then configured for eTPU operation

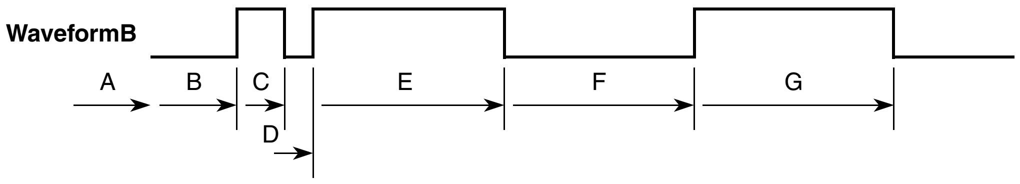
Three basic waveforms are used throughout this example;

WaveformA (as defined by my_QOM_event_arrayA; used to build the event table used by the QOM function):



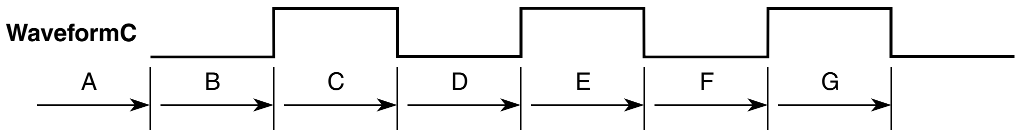
A	Pin High After 0x010000 TCR1 Counts
B	Pin Low After 0x008000 TCR1 Counts
C	Pin High After 0x004000 TCR1 Counts
D	Pin Low After 0x002000 TCR1 Counts
E	Pin High After 0x010000 TCR1 Counts
F	Pin Low After 0x010000 TCR1 Counts
G	Pin High After 0x010000 TCR1 Counts

WaveformB (as defined by my_QOM_event_arrayB; used to build the event table used by the QOM function):



A	Pin Low After 0x010000 TCR1 Counts
B	Pin High After 0x008000 TCR1 Counts
C	Pin Low After 0x004000 TCR1 Counts
D	Pin High After 0x002000 TCR1 Counts
E	Pin Low After 0x010000 TCR1 Counts
F	Pin High After 0x010000 TCR1 Counts
G	Pin Low After 0x010000 TCR1 Counts

WaveformC (as defined by my_QOM_event_arrayC; used to build the event table used by the QOM function):



A	Pin Low After 0x010000 TCR1 Counts
B	Pin High After 0x010000 TCR1 Counts
C	Pin Low After 0x010000 TCR1 Counts
D	Pin High After 0x010000 TCR1 Counts
E	Pin Low After 0x010000 TCR1 Counts
F	Pin High After 0x010000 TCR1 Counts
G	Pin Low After 0x010000 TCR1 Counts

4.2.3 Channel Operation

4.2.3.1 QOM0 (ETPUA0)

QOM0 generates waveformA in single shot operation mode using the TCR1 timebase and middle priority. The initial pin state is low.

4.2.3.2 QOM1 (ETPUA1)

Before initializing QOM1/QOM2, a delay is deliberately introduced by the host. This is to demonstrate the difference between immediate reference mode and address reference mode.

QOM1 uses single shot operation mode and address reference mode. In address reference mode, the first event is scheduled relative to a value stored in an address pointed to by a reference address pointer. In this case, the host assigns the reference address pointer to be the address of the Last_Match_Time variable on QOM0. Consequently, the first match on QOM1 is referenced to QOM0's last event.

QOM1 is initialized to generate waveformB. The timebase is TCR1 and the pin is initialized high. The channel is assigned middle priority.

4.2.3.3 QOM2 (ETPUA2)

QOM2 uses continuous operation mode and immediate reference mode.

In immediate reference mode, the first event is scheduled with respect to the current TCR value. QOM2 is initialized to generate waveformB. The timebase is TCR1 and the pin is initialized high. The channel is assigned middle priority.

4.2.3.4 QOM3 (ETPUA6)

QOM3 demonstrates operation of a channel which has been configured for link operation. The channel is configured for single shot operation mode. The channel is configured to receive a link. On receipt of the link, the channel outputs waveformB with an immediate reference using timebase TCR1. After the link has been sent, the host waits for the interrupt service request from the QOM3 channel. The host then reconfigures the channel with new data so that waveformC is generated with the first event being referenced to the last event in waveformB. A more sophisticated scheme using the host interrupt handling capabilities could be used to transmit additional new data from a QOM channel.

4.2.3.5 QOM4 (ETPUA10)

QOM4 demonstrates operation of a channel which has been configured for link operation in loop operation mode. On receipt of the link, the channel outputs waveformB with an immediate reference using timebase TCR1. Before the loops have been completed another link is sent to the channel. The channel re-starts the event table at the beginning. The number of wholly executed loops remains as requested. i.e. the loop counter is not reset to its initially requested value.

The host waits until QOM4 has finished all the programmed loops. Another link is then sent and the channel remains inactive. This demonstrates that in order for the channel to be able to service a new link after completing the programmed loops, it must be reinitialized.

4.2.3.6 QOM5 (ETPUA14)

QOM5 demonstrates a channel which has been configured for link operation in continuous operation mode. On receipt of the link, the channel outputs waveformB continuously with an immediate reference using timebase TCR1. Another link is sent to the channel. The channel restarts the event table at the beginning and resumes outputting the waveform as before.

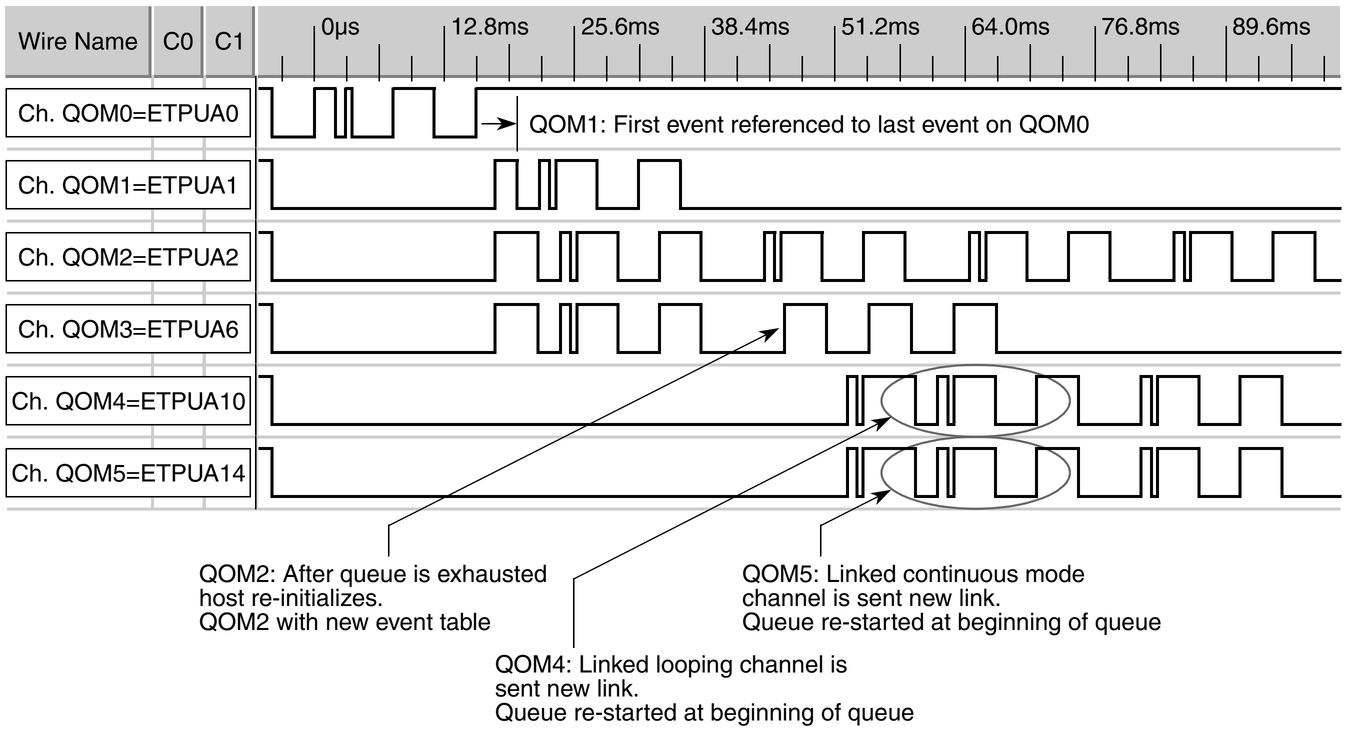


Figure 10. qom_example2.c Output

5 Summary and Conclusion

This eTPU QOM application note provides the user with a description of the Queued Output Match eTPU function usage and with examples. The simple C interface routines to the QOM eTPU function enable easy implementation of the QOM function in applications. The functions are targeted for the MPC5500 family of devices, but they can be used with any device that has an eTPU.

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

AN2857
Rev. 0
11/2004