intel®

AP-449

APPLICATION NOTE

A Comparison of the Event Processor Array (EPA) and High Speed Input/Output (HSIO) Unit

BRIAN HINTZMAN
AUTOMOTIVE APPLICATIONS ENGINEER

March 1991

Order Number: 270968-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

A Comparison of the Event Processor Array (EPA) and High Speed Input/Output (HSIO) Unit

CONTENTS

CONTENTS

1.0 INTRODUCTION

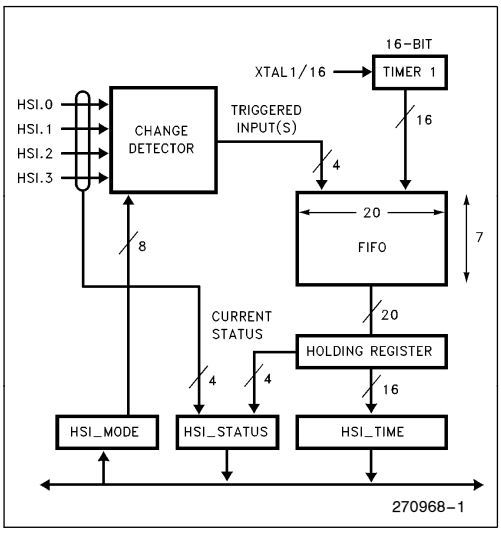
In modern control applications, high speed communications with the outside world is an essential feature of today's microcontroller. To handle such signals, Intel's new generation of 16-bit microcontrollers, starting with the 8XC196KR, offers the Event Processor Array (EPA). The EPA's many abilities make it versatile and ideal for such high speed signals.

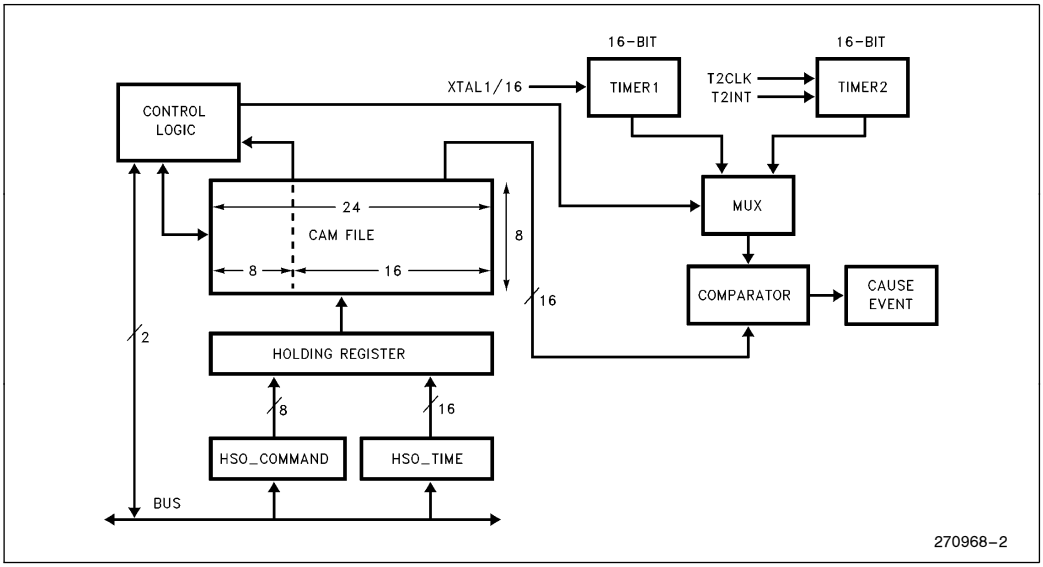
Intel's 16-bit microcontrollers prior to the 8XC196KR device provide the High Speed Input/Output unit (HSI/O) for high speed event control. The HSI/O serves as a basis for all high speed communications modules on Intel microcontrollers. The EPA is the result of an evolutionary process evaluating the needs of high speed input and output control. Because the EPA is a new peripheral module of the 16-bit microcontroller, the immediate question is how this module is comparable to its counterpart on previous MCS-96 devices. Though on the surface the HSI/O and EPA modules may appear to be vastly different, the reality is that the two are functionally very similar.

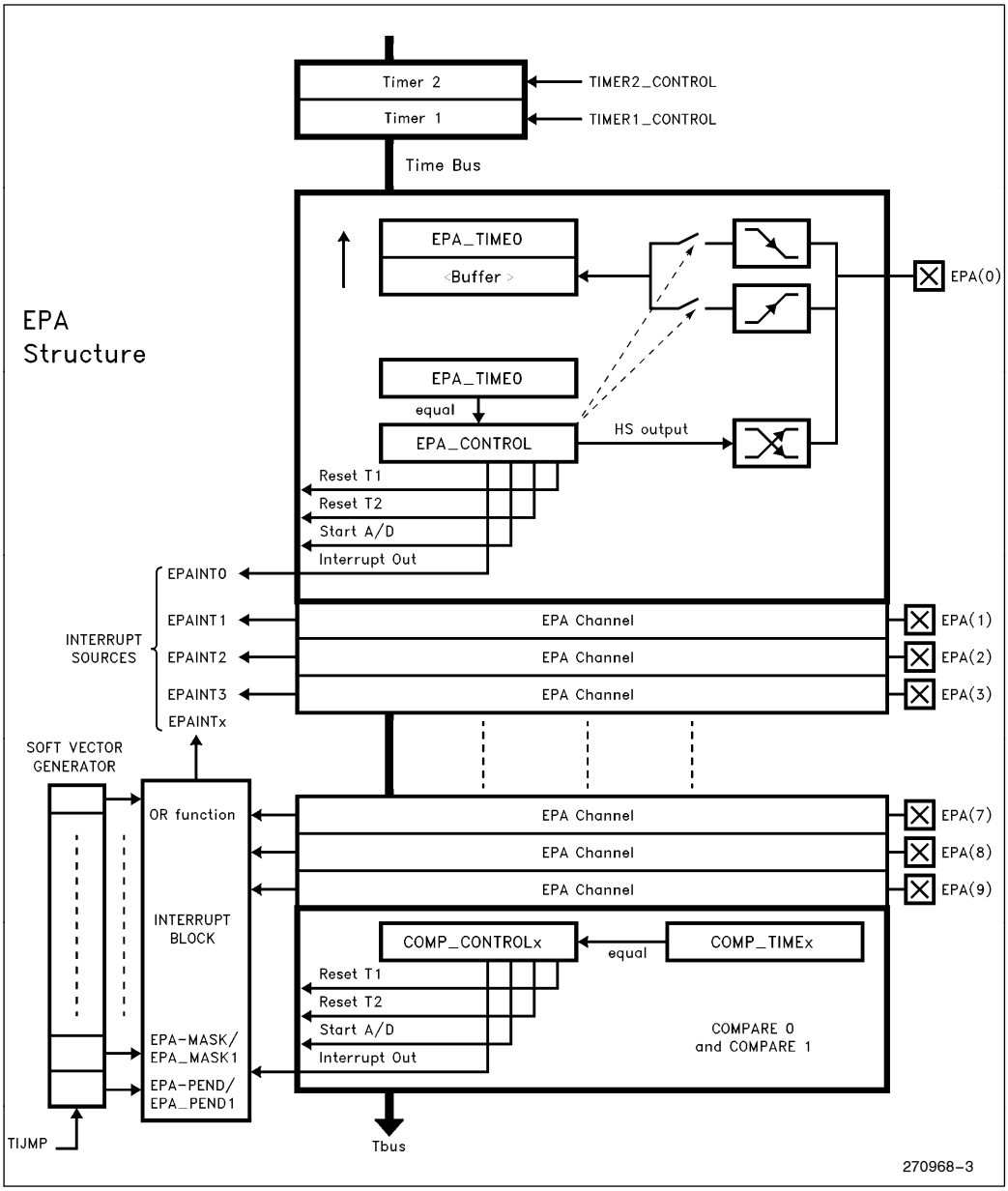
In this application note a general familiarity with the MCS-96 architecture is assumed. Its purpose is to compare the functions of the HSI/O and EPA and assist the programmer who wishes to use the EPA module and is already familiar with the HSI/O.

1.1 General Overview

1.1.1 HSI/O






EPA
Structure
Timer 2
Timer 1
TIMER2_CONTROL
TIMER1_CONTROL
Time Bus
EPA_TIME0
<Buffer >
EPA(0)
EPA_TIME0
equal
EPA_CONTROL
HS output
Reset T1
Reset T2
Start A/D
Interrupt Out
EPAINT0
EPAINT1
EPAINT2
EPAINT3
EPAINTx
INTERRUPT
SOURCES
EPA Channel
EPA(1)
EPA(2)
EPA(3)
SOFT VECTOR
GENERATOR
OR function
INTERRUPT
BLOCK
EPA(7)
EPA(8)
EPA(9)
COMP_CONTROLx
equal
COMP_TIMEx
EPA-MASK/
EPA_MASK1
EPA-PEND/
EPA_PEND1
COMPARE 0
and COMPARE 1
TIJMP
Tbus
270968-3

The HSI/O is capable of handling timed input and output events on a number of exterior pins. There are two dedicated input pins, four dedicated outputs, and two pins that are multiplexed between input and output. The module has two internal timers for the timing and scheduling of events. Up to a total of eight incoming events, rising or falling edges, on all input pins can be buffered at once in a FIFO storage unit. Up to eight output events for all output pins can be buffered at once in the Content Addressable Memory (CAM) unit. The HSI/O also features either one or three Pulse Width Modulation channels. The HSI/O appeared on the original 8X9X device and has changed only slightly as new parts have been released. Further information on the specific differences between these devices is given below. Also see the User's Guides for the 8096BH, 8XC196KB, and 8XC196KC in the *16-Bit Embedded Controllers Handbook.*

1.1.2 EPA

The EPA had many attributes similar to the HSI/O. In the case of the 8XC196KR, there are ten EPA channels, called capture/compare modules, each of which can be selected as input or output. In addition there are two compare channels dedicated to the timing of internal events only. Again the module has two internal timers for the timing and scheduling of events. Two input events are buffered separately for each pin configured as an input and one output event for each pin configured as an output. More detailed information on all these functions is provided below as well as in the 8XC196KR User's Guide in the *Automotive Products Handbook.*

1.1.3 PTS

An additional module, first introduced on the 8XC196KC and 8XC196KR, called the Peripheral Transaction Server (PTS) greatly enhances the capabilities of the EPA. The PTS is an interrupt handler that performs very fast Direct Memory Access interrupts. These interrupts can be initiated by internal or external events, hence the close connection between the EPA and PTS. Though the PTS will not specifically be discussed here, more information can be obtained from the 8XC196KC and 8XC196KR User's Guide. Examples of using the PTS in conjunction with the EPA are given through this text.

1.1.4 HOW TO USE THIS APPLICATION NOTE

The purpose of this paper is to provide a direct, side by side comparison of the HSI/O module in the left-hand column and the EPA module in the right-hand column. Both accomplish nearly the same functions but their implementations, and therefore for the necessary software, can be quite different. Specific functions of HSI/O and EPA are discussed and, where applicable, actual code showing actions performed in both modules is provided. In some cases, examples are also given of functions that have no equivalent in the other module.

Most of the code examples for the EPA make use of the new windowing capabilities as well as the new port configuration techniques that, like the EPA itself, are new on the 8XC196KR device. The window and port functions are not described here but are discussed in the 8XC196KR User's Guide. Note that the new windowing allows many addresses outside of the 00 to FF address space to be referenced as registers. The equate statements defining the register names used in this text are defined in the appendix. Also, whenever actual times are referred to instead of state times, a 16 MHz clock frequency is assumed. Finally, where the letters BH, KB, KC, and KR appear, these refer to the 8X9XBH, 8XC196KB, 8XC196KC, and 8XC196KR respectively.

2.0 TIMERS

HSI/O

2.1a Clocking

Both TIMER1 and TIMER2 are 16-bit timers used for time stamping of incoming events and for scheduling of output events. TIMER1 is a free running timer whose only clocking mode is internal clocking once every eight state times. TIMER2 is clocked by transitions, both rising and falling edges, on either T2CLK or HSI.1 but the maximum clock rate is still once every eight state times. On the KB and KC, TIMER2 can be clocked once every state time in Fast Increment Mode and on the KC, TIMER2 can also be clocked internally.

```
LDB     IOC0, #00H
LDB     IOC1, #00H
LDB     IOC2, #00H
```

- T2CLK pin is clock source
- Count up clocked by rising and falling edges on T2CLK pin
- External reset disabled
- Do not reset timer each write
- Disable overflow interrupt
- Disable fast increment mode - KB and KC only

Ex. 1a. Configure TIMER2 to Count Externally

EPA

2.1b Clocking

Both TIMER1 and TIMER2 are 16-bit timers. TIMER1 and TIMER2 can be clocked externally through T1CLK and T2CLK or clocked internally. The maximum clock rate, internal or external, is the chip clock rate divided by 4. Internally, prescaling allows clocking at several selectable fractions of the chip clock frequency. Externally, prescaling allows clocking at several selectable fractions of the T1CLK or T2CLK frequency. TIMER1 and TIMER2 can be chained together to produce a single 32-bit counter by clocking TIMER2 with the overflow of TIMER1. A quadrature counting option allows use of the EPA with an encoder wheel.

The principle difference between the EPA and HSI/O timers is that TIMER1 is not free running, it MUST be configured in order to count whereas it formerly would automatically begin counting on start up. Also note that all functions of the timers are held in the two TIMERn__CONTROL registers rather than mixed in with the IOCn registers allowing complete configuration2 of a timer in one memory write.

```
LDB     WSR, #7EH
LDB     P1IO__2, #0FFH
LDB     P1SSEL__2, #01H
LDB     WSR, #7CH
LDB     TIMER2__CONTROL__2, #0C8H
```

- P1.0 selected for input as T2CLK pin
- Count up clocked by rising and falling edges on T2CLK pin

Ex. 1b. Configure TIMER2 to Count Externally

```
LDB     WSR, #7CH
LDB     TIMER1__CONTROL__2, #0C2H
LDB     TIMER2__CONTROL__2, #0F0H
```

- TIMER1 counts up with 1 μs period
- TIMER2 counts on TIMER1 overflow and in the same direction as TIMER1

Ex. 1c. Chaining Two Timers Together

HSI/O

2.2a Reset

TIMER1 can only be reset by a chip reset. TIMER2 can be reset by a chip reset, setting bit 1 in IOC0, setting either T2RST or HSI.0 depending on the value of IOC0.5, or by any of the HSO's. On the KB and KC, both timers may be written with any value, which may also be considered "resetting" the timer.

```
LDB     HSO_COMMAND, #8EH
LD      HSO_TIME, #500
```

- Resets TIMER1 every 500 clocks
- Event locked in CAM (KB and KC only)
- No interrupts generated by CAM or TIMER1

Ex. 2a. Reset TIMER2 with HSO

EPA

2.2b Reset

Both timers can be reset by any of the EPA channels. Choosing the appropriate EPA control mode allows either timer to reset itself or the opposite timer. Both timers can be loaded with any time value. Though there is no auto-reload capability this function can be produced using the PTS (see Appendix 7.3).

The main difference here is that the new system has no dedicated T2RST pin for external resetting of TIMER2. However, any of the 10 EPA channels can be configured to reset either timer or an input event as shown below.

```
LDB     WSR, #7CH
LD      COMP_CONTROL0_2, #0FE49H
LD      COMP_TIME0_2, #500
LDB     TIMER1_CONTROL_2, #0C2H
```

- TIMER1 counts every 1 μs
- Resets TIMER1 every 500 clocks (500 μs)
- COMP0 time entry locked
- No interrupts generated by COMP0 or TIMER1

Ex. 2b. Reset TIMER2 with Compare Channel 0

```
LDB WSR, #7EH
LDB P1IO_2, #0FFH
LDB P1SSEL_2, #01H
LDB WSR, #7BH
LDB EPA_CONTROL0_2, #0FE23H
LDB WSR, #7CH
LDB TIMER2_CONTROL_2, #0C2H
```

- Port 1.0 is EPA input
- Captures on rising edge
- TIMER2 counts every 1 μs
- Captures TIMER1, resets TIMER2, (opposite timer)
- NOTE: MUST capture one timer and reset the OPPOSITE timer to reset on input event
- NOTE: Cannot be done without interrupts on HSI/O

Ex. 2c. Reset TIMER2 on Input Rising Edge

HSI/O

2.3a Count Direction

In the BH part, both timers can only count up. On the KB and KC, the TIMER2 direction can be chosen by the value of Port 2.6 The timer direction cannot be controlled internally.

```
LDB     IOC2, #02H
```

- TIMER2 counts down if Port 2.6 is high, up if low

Ex. 3a. Control TIMER2 Direction Externally

EPA

2.3b Count Direction

Both timers can determine their count direction from either their internal timer control register or on the T1DIR and T2DIR pins. Also, the count direction of TIMER2 can be set to match that of TIMER1 so that TIMER1 and TIMER2 together perform 32-bit up/down counting.

Additionally, because the T2DIR pin is the same as the EPA2 pin, T2DIR can be controlled directly using EPA2. Setting the T2DIR as the direction control for TIMER2 and configuring EPA2 as an output allows the value of EPA2 to control the count direction of TIMER2 (see Ex. 3c.)

```
LDB     WSR, #7CH
LDB     P1IO__2, #0FFH
LDB     P1SSEL__2, #04H
LDB     TIMER2__CONTROL__2, #90H
```

- P1.2 selected as T2DIR
- TIMER2 counts up if T2DIR is high, counts down if low

Ex. 3b. Control TIMER2 Direction Externally

```
LDB     WSR, #7EH
LDB     P1REG__2, #0FBH
LDB     P1IO__2, #0FBH
LDB     P1SSEL__2, #04H
LDB     WSR, #7BH
LD      EPA__CONTROL2__2, #0FE70H
LDB     WSR, #7CH
LDB     TIMER1__CONTROL__2, #0C2H
LDB     TIMER2__CONTROL__2, #0C6H
```

- P1.2 selected as EPA2 input and T2DIR
- EPA2 toggles pin so it changes TIMER1 direction on EPA__TIME2 match
- NOTE: Time entry not locked here so a new EPA__TIME2 value must be written for each direction change

Ex. 3c. Control TIMER2 Direction Internally with EPA2

3.0 INPUTS

HSI/O

3.1a Input Channels

The HSI/O has a total of four input channels. Two are dedicated lines and two are selectable as input or output. All events recorded on these lines go directly to the FIFO unit.

3.2a Input Capture

The HSI records the time (value of TIMER1 only) of logic transitions along with the pin on which they occur in the FIFO structure. Up to eight events can be stored in the FIFO at one time. The HSI is capable of capturing events on rising edges only, falling edges only, rising and falling edges, or every 8th rising edge. Events on separate pins that occur within the same clock period are stored in the same FIFO entry. Additionally, on the KB and KC, rising edges on the P2.7 pin capture the value of TIMER2 in the T2CAP register for input event processing that bypasses the FIFO and creates its own interrupt.

```
CLRB    INT_PEND
LDB     IOC0, #10H
LDB     IOC1, #00H
LDB     HSI_MODE, #10H
LDB     INT_MASK, #04H
EI
```

- HSI.2 input enabled
- Captures each positive edge on HSI.2
- Tags event with TIMER1 time (no choice)
- HSI interrupts on Holding Register loaded

Ex. 4a. Choosing HSI.2 as Input

EPA

3.1b Input Channels

The EPA has 10 external pins, each of which can individually be selected for input capture. Unlike the HSI/O, the EPA pins also serve as port pins. This means the pins MUST first be configured to function for the EPA and then the EPA_CONTROLn register must also be configured for capturing the appropriate input event. See the 8XC196KR User's Guide for more information on configuring the ports.

3.2b Input Capture

The EPA can time stamp input events with the time value of TIMER1 and TIMER2. In the EPA, each channel stores its input events separately in its EPA_TIMEn register rather than mixed together as in the FIFO. Each time register is buffered allowing the storage of two input events at once. Input capture events include rising edges, falling edges, or rising and falling edges. Capture on every 8 rising edges as the HSI/O does can be handled by the EPA using the PTS with more flexibility (see Section 7.1). The T2CAP register of the KB and KC parts does not exist in the EPA because every channel is capable of performing this function and generating its own interrupt.

```
CLRB    INT_PEND
LDB     WSR, #7EH
LDB     P1REG_2, #0FFH
LDB     P1IO_2, #0FFH
LDB     P1SSEL_2, #04H
LDB     WSR, #7BH
LD      EPA_CONTROL2_2, #0FE20H
LDB     INT_MASK, #04H
EI
```

- P1.2 selected as EPA input
- Captures each rising edge
- Tags event with TIMER1 time
- EPA2 interrupts on capturing rising edge

Ex. 4b. Choosing EPA.2 as Input

HSI/O

3.3a Interrupts

All incoming events produce interrupts solely through the FIFO unit and vector through the same address in the vector table (except for TIMER2 capture on the KB and KC). Since the time of an input event from ANY pin is provided in the HSI__TIME register, this register MUST be read in the interrupt service routine to allow further interrupts from any HSI pin to occur. Also, if multiple HSI channels are used, because of the common FIFO storage unit, the interrupt service routine must decode the HSI__STATUS register to determine which channel created the interrupt.

Because of the storage space of the FIFO, several input events can be stored at once before an interrupt is required. On the BH part an interrupt can be created with one or six entries in the FIFO. With the KB and KC parts an interrupt can also be created on the fourth entry, or when the FIFO is half full.

EPA

3.3b Interrupts

Unlike the HSI/O where all input interrupts are vectored to the same interrupt service routine, several of the EPA channels generate their own interrupts directly with no decoding. EPA channels 0 through 3 each have their own bits in the INT__PEND register and their own vectors allowing easy interrupt handling for these channels. EPA channels 4 through 9 and Compare channels 0 and 1 all generate interrupts to the EPAINTX bit in the INT__PEND register. However, for these interrupts, the TIJMP command has been added to the 8XC196KR instruction set to be used with the EPAIPV register allowing the use of individual interrupt routines for EVERY interrupt with very little decode overhead in determining the source of the interrupt (see Section 7.5). Similar to the HSI/O, however, the EPA__TIMEn register must be read during each interrupt service routine to allow further interrupts on that channel.

The addition of the PTS module makes the interrupt capability of the EPA even more flexible. For every interrupt produced by the EPA, it is possible to produce a PTS interrupt instead. This is done by setting the corresponding bit in the PTS__SELECT register. As long as this bit is set, all interrupts associated with this bit will be PTS interrupts. Again, see the 8XC196KR User's Guide for information on using the PTS.

```
CLRB    INT__PEND
LDB     WSR, #7EH
LDB     P1REG__2, #0FFH
LDB     P1IO__2, #0FFH
LDB     P1SSEL__2, #04H
LDB     WSR, #7BH
LD      EPA__CONTROL2__2, #0FE20H
LD      PTS__SELECT, #0004H
LDB     INT__MASK, #04H
EPTS
EI
```

- Same as previous example except uses PTS interrupt

Ex. 4c. Choosing EPA2 as Input, Using PTS to Service Interrupt

HSI/O

3.4a Reading HSI Current State

The current state of any of the HSI pins can be read directly from the HSI__STATUS register. However, there are two difficulties in handling HSI interrupts.

The first is that reading the HSI__TIME register causes the next entry to be read out of the FIFO and written to the HSI__STATUS register. If two or more HSI events occurred in the same clock cycle then handling the first event will cause the HSI__STATUS register to be written over and the second event is lost. This is because all of the individual HSI interrupt bits are stored in the HSI__STATUS register. This means an extra copy of HSI__STATUS must be held in software to be sure no events are lost.

The second difficulty is that the IOS1 register is cleared every time it is read. The IOS1 register is used to indicate when the HSI Holding Register is full, when the FIFO is full, and when any of the software timers have expired. This means that an extra copy of this register also must be kept in software if one wishes to check the status of all conditions. Great care must be taken when writing software for both situations.

EPA

3.4b Reading EPA Current State

Determining the current state of any of the EPA pins can be accomplished easily by reading either the P1PIN register (EPA 0 through 7) or the P6PIN register (EPA 8 and 9). Reading bits to decode which input channel caused an interrupt is not necessary as in the HSI/O because the decoding is performed by hardware. As stated in the previous section, EPA0 through 3 have their own interrupt bits and vectors. All other EPA channels are vectored through the EPAINTX vector. Nowhere is it necessary to keep extra copies of registers to avoid losing interrupt data.

4.0 OUTPUTS

HSI/O

4.1a Output Channels

Four dedicated output lines are available to the HSI/O as well as the two multiplexed lines that are selectable as input or output. Output events to all six output lines are controlled solely by the CAM unit. While input events can only be time stamped by TIMER1, output events can be based on TIMER1 or TIMER2.

```
LDB     HSO__COMMAND, #20H
ADD     HSO__TIME, TIMER1, #200
```

- HSO.0 sets pin in 200 clocks of TIMER1
- TIMER1 counts every 8 state times (no choice)

Ex. 5a. Set HSO.0 Pin Based on TIMER1

EPA

4.1b Output Channels

All ten of the external EPA pins can individually be configured for output events. Each channel has its own EPA__CONTROLn and EPA__TIMEn register for the scheduling of events. EPA channels 0 and 1 and channels 2 and 3 can be "mapped" together such that all of their external events appear on EPA pins 1 and 3 respectively. Compare channels 0 and 1, though they do not seem to have external pins, can be configured to set or reset pins. Output events of Compare channel 0 appear on EPA8 and those of Compare channel 1 appear on EPA9 (EPA8/Compare0 and EPA9/Compare1 can be thought of as permanently mapped together). The remapping of channels allows two events to be scheduled to occur on one pin in quick succession, faster than could be done with interrupts. Note: the mapping of two channels together on one pin is for output events only. This does not work for input.

```
LDB     WSR, #7EH
LDB     P1REG__2, #0FEH
LDB     P1IO__2, #0FEH
LDB     P1SSEL__2, #01H
LDB     WSR, #7BH
LD      EPA__CONTROL0__2, #0FE60H
LD      EPA__TIME0__2, #200
LDB     WSR, #7CH
LDB     TIMER1__CONTROL__2, #0C2H
```

- P1.0 selected as EPA, output, initially low
- EPA0 sets pin in 200 clocks of TIMER1
- TIMER1 counts every 1 μs

Ex. 5b. Set EPA0 Pin Based on TIMER1

HSI/O

4.2a Output Events

Output events are initiated immediately when the time tag of an output event matches the value on the selected timer. The HSI/O can produce a number of different "output" events, not all affecting the external pins. The unit can raise or lower the logic levels of one or a group of the pins, reset TIMER2, or initiate an A/D conversion, while each action can also optionally cause an interrupt. Using the 4 software timers the unit can cause an interrupt without taking any other action. When several software timers are used, however, the source of interrupt must be decoded because all timers use the same interrupt vector.

```
CLRB    INT__PEND
LDB     HSO__COMMAND, #58H
ADD     HSO__TIME, TIMER2, #1000
LDB     INT__MASK, #04H
```

- Software timer 0 set to interrupt in 1000 clocks of TIMER2

*NOTE: Decoding of interrupt source by reading IOS1

Ex. 6a. Use Software TIMER0 to Create Interrupt

EPA

4.2b Output Events

EPA output events can be initiated from a time tag match with either timer. The various events controlled are similar to those of the HSI/O including setting or resetting a pin or toggling the pin, resetting either timer, and starting an A/D conversion, where each event can be accompanied by an interrupt. The four software timers of the HSI/O are replaced in the EPA by simply allowing an interrupt to occur without performing another action.

```
CLRB    INT__PEND
LDB     WSR, #7CH
LD      COMP__CONTROL0__2,
        #0FE80H
ADD     COMP__TIME0__2, TIMER2,
        #1000
LDB     INT__MASK, #01H
EI
```

- COMP0 set to interrupt in 1000 clocks of TIMER2
- NOTE: TIJMP along with EPAIPV must be used to get to interrupt service routine

Ex. 6b. Use COMP0 to Create Interrupt

```
LDB     WSR, #7EH
LDB     P1REG__2, #0FDH
LDB     P1IO__2, #0FDH
LDB     P1SSEL__2, #02H
LDB     WSR, #7BH
LD      EPA__CONTROL0__2, #0FE60H
LD      EPA__TIME0__2, #500
LD      EPA__CONTROL1__2, #0FF50H
LD      EPA__TIME1__2, #501
LDB     WSR, #7CH
LDB     TIMER1__CONTROL__2, #0C0
```

- EPA0 and 1 remapped together
- P1.1 selected as EPA output
- EPA0 sets pin in 500 clocks of TIMER1, EPA1 reset pin 1 clock later
- TIMER1 counts every 250 ns

Ex. 6c. Using Remap Function to Produce Two Events on One Pin

HSI/O

4.3a Output Event Buffering (CAM)

The CAM unit allows up to eight output events to be written, or buffered, at once. Each event occurs when its time tag matches the corresponding timer value, regardless of the order events are written to the CAM.

The CAM is loaded by writing the event to the HSO__COMMAND register followed by writing the time to the HSO__TIME register. When the HSO__TIME register is written it may take up to eight state times to actually load the CAM. When writing multiple events to the CAM one must be careful to observe the eight state time limit in writing the HSO__TIME register.

Once an event is written to the CAM it cannot be removed if, for instance, it needs to be rescheduled. For commands that affect the external pins, writing the opposite action for the same time and pin will prevent any action from being taken while taking up two of the eight CAM entries. For other actions, such as resetting TIMER2, there is no appropriate "opposite" action. However, the entire CAM can be cleared to remove the event by setting IOC2.7 (not always a viable option).

4.4a Event Locking

On the KB and KC parts, events written to the CAM can be "locked". These events remain in the CAM instead of disappearing after executing once as normally occurs. They occur EVERY time the clock counts back around to match the time tag of the event. Care must be taken when using this option as these events can only be cleared from the CAM by clearing the entire CAM (setting bit IOC2.7) or resetting the chip.

```
LDB     HSO__COMMAND, #0A0H
ADD     HSO__TIME, TIMER1, #200
```

- Same as Ex. 5a but event is now locked in CAM

Ex. 7a. Lock Event in CAM

EPA

4.3b Output Event Buffering

One of the main differences between the HSI/O and EPA structures is that the EPA has no CAM unit. Instead, each pin holds one timed output event in the EPA__ TIMEn register.

An EPA output event is set up by first writing to the EPA__CONTROLn register for the desired channel and then writing the EPA__TIMEn register. As soon as the event occurs, normal software or and interrupt routine can set up the next timed event.

Since all of the EPA channels have their own separate EPA__CONTROLn registers, events on different channels can be written as fast as desired. The eight state time wait period does not have to be observed as in the HSI/O. If a pending event must be changed or rescheduled, the appropriate register, either control or time, simply needs to be written over before the event takes place. Clearing all pending events is never necessary as is sometimes the case in the HSI/O.

The one advantage lost in the EPA is the scheduling of up to eight events at once as in the CAM. However, the fast interrupts of the PTS can be used to schedule many new events as soon as the previous event occurs. Also many more then eight events can be scheduled using this technique. Please see Appendix for an example.

4.4b Event Locking

The EPA also has locked events that occur every time the time tag matches the appropriate timer. Unlike the HSI/O they are easily removed when they are no longer desired. Simply writing the EPA__CONTROLX register with a new value will enable a new event that may or may not be locked, depending on the value of the RE bit.

```
LD      EPA__CONTROL0__2, #0FE68H
LD      EPA__TIME0__2, #200
```

- Same as Ex. 5b but event is now locked in EPA__TIME0 register

Ex. 7b. Lock Event in EPA__TIMEn

5.0 PULSE WIDTH MODULATION OUTPUT (PWM)

HSI/O

5.1a Output Channels

Another feature of Intel 16-bit microcontrollers prior to the KR device, yet functionally separate from the HSI/O structure, is the Pulse Width Modulation Output or PWM. Intended for slow response analog devices such as meters and motors, the PWM acts as a D/A converter producing a constant frequency, variable duty cycle square pulse train. The unit has its own eight-bit counter. An eight-bit PWM__CONTROL register determines on which of the 256 counts the output goes low. The BH part has one such dedicated channel while the KB and KC parts have 3 PWM pins.

5.2a Frequency

On the BH part, the PWM frequency is set where one period is equal ot 256 state times. On the KB and KC parts the period can equal 256 or 512 state times. The actual frequency is then completely determined by the chip clock frequency.

5.3a Duty Cycle

The PWM duty cycle is only selectable with eight bits of precision, it must be one of 256 discrete values. Even when the period is equal to 512 state times there is no gain in duty cycle resolution, only a change in PWM frequency. The duty cycle can be of either polarity as long as care is taken to note that time in the PWM__CONTROL register is the time the signal goes LOW. Also note that a 0% duty cycle (always low) is achievable while 99.6% duty cycle (high) is the maximum.

```
LDB     IOC1, #01H
LDB     PWM__CONTROL, #80H
```

- PWM with 50% duty cycle

Ex. 8. Typical Use of PWM Unit

EPA

5.1b Output Channels

The KR device has no PWM output unit. The EPA by itself or using the PTS performs the functions of a dedicated PWM unit. Using the PTS PWM Mode or PTS PWM Toggle Mode the user is able to produce any signal the PWM unit produces with much more control over the actual shape of the wave. A variety of PWM outputs can be used on any of the 10 EPA channels. A number of those possible are shown in complete program examples in Section 6.

5.2b Frequency

Where the HSI/O's PWM unit gives the user at most two frequencies to choose from, the EPA and PTS combination provides a very wide range of possible output frequencies. Depending on the process used a period range of as little as two state times up to 65536 state times (or longer) is possible. (Again, see Section 6 for examples.)

5.3b Duty Cycle

Any duty cycle of either polarity is achievable using the appropriate example found in Section 6. The resolution in selecting the duty cycle is always 1 state time. The number of different duty cycles is only limited by the number of state times in the period of the PWM signal produced.

6.0 EPA PWM EXAMPLES

6.1 Example # 1

The first example stores and locks a time in an EPA__TIMEn register while the EPA__CONTROLn register instructs the EPA channel to toggle the pin. Every time the clock counts around to the time value in the EPA__TIMEn register the pin toggles. This makes a low frequency, 50% duty cycle square wave. The frequency is variable by changing the TIMER1 prescaling. No CPU overhead required.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE (KR.INC)
$LIST

STK       EQU    200H

CSEG AT 2080H

      LD      SP,#STK                 ; Initialize Stack Pointer
      CLRB    INT_PEND                ; Clear out interrupts
      CLRB    INT_PEND1               ;

      LDB     WSR,#7EH                ; 32 byte window on 1FC0H
      LDB     P1REG_2,#0FFH           ; Turn off pull down
      LDB     P1IO_2,#0FEH            ; P1.0 is output
      LDB     P1SSEL_2,#01H           ; P1.0 is EPA0

      LDB     WSR,#7BH                ; 32 byte window on 1F60H
      LD      EPA_CONTROL0_2,#0FE78H  ; TIMER1, toggle output, lock time entry
      LD      EPA_TIME0_2,#100        ; Time tag to toggle output

      LDB     WSR,#7CH                ; 32 byte window on 1F80H
      LD      TIMER1_CONTROL_2,#0C2H  ; Enable timer, count up, 1us period

SELF:
      SJMP    SELF                    ; Let EPA take over

END
```

270968-4

6.2 Example #2

Here the PTS is used to produce a square wave of 50% duty cycle where the selection of frequency is much greater than in the previous example. Using the PTS PWM Mode, the interrupt adds the value of CONST1 to the value in the EPA__TIME0 register. On each interrupt the external pin is toggled. The PWM frequency is then selectable with 16 bits of resolution. This example produces a 50% duty cycle 1 KHz square wave. Very high frequency waves cannot be made, however, because two interrupt latencies will occur during each period.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

STK     EQU    200H

CSEG AT 2048H
       DCW          PTS_UNUSED0

CSEG AT 2008H
       DCW          EPA0_ISR

RSEG AT 70H
       PTS_UNUSED0:          DSB    1
       PTS_CONTROL0:         DSB    1
       PTS_SRC0:             DSW    1
       CONST1:               DSW    1

CSEG AT 2080H
       DI
       DPTS

       LD      SP,#STK                 ; Initialize Stack Pointer
       CLRB    INT_PEND                ; Clear out interrupts
       CLRB    INT_PEND1

       LDB     PTS_CONTROL0,#40H       ; PTS PWM Mode
       LD      PTS_SRC0,#EPA_TIME0     ; Source is EPA_TIME0
       LD      CONST1,#500             ; Constant for half period

       LDB     WSR,#7EH                ; 32 byte window on 1FC0H
       LDB     P1REG_2,#0FFH           ; Turn off pull down
       LDB     P1IO_2,#0FEH            ; P1.0 is output
       LDB     P1SSEL_2,#01H           ; P1.0 is EPA0

       LDB     WSR,#7BH                ; 32 byte window on 1F60H
       LD      EPA_CONTROL0_2,#0FE70H  ; TIMER1, toggle output
       LD      EPA_TIME0_2,#100        ; Initial toggle time

       LDB     WSR,#7CH                ; 32 byte window on 1F80H
       LD      TIMER1_CONTROL_2,#0C2H  ; Enable timer, 1us period

       LD      PTS_SELECT,#0010H       ; Enable PTS interrupt for EPA0
       LDB     INT_MASK,#10H           ; Enable EPA0 interrupts

       EPTS
       EI

SELF:
       SJMP    SELF                    ; Let EPA take over

EPA0_ISR:
       ORB     INT_PEND,#10H           ; Select PTS interrupt
       RET

END
```

270968-5

6.3 Example #3

The PTS Toggle Mode is used to produce a square wave with other than 50% duty cycle using only one EPA channel. On each interrupt either CONST1 or CONST2 is added to the EPA__TIME0 register. On the following interrupt the other constant is added. Each time the external pin is toggled. This allows specifying the high time of the signal in one constant and the low time in the other. The period of the signal is the sum of CONST1 and CONST2, the high time and the low time. This sum must be greater than the maximum interrupt latency.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

STK     EQU    200H

CSEG AT 2048H
       DCW          PTS_UNUSED0

CSEG AT 2008H
       DCW          EPA0_ISR

RSEG AT 70H
       PTS_UNUSED0:    DSB 1
       PTS_CONTROL0:   DSB 1
       PTS_SRC0:       DSW 1
       PTS_CONST0:     DSW 1
       PTS_CONST1:     DSW 1

CSEG AT 2080H
       DI
       DPTS

       LD      SP,#STK                 ; Initialize stack
       CLRB    INT_PEND                ; Clear out interrupts
       CLRB    INT_PEND1               ;

       LDB     PTS_CONTROL0,#42H       ; PTS Toggle Mode
       LD      PTS_SRC0,#EPA_TIME0     ; Source is EPA_TIME0
       LD      PTS_CONST0,#500         ; Constant for positive half cycle
       LD      PTS_CONST1,#800         ; Constant for negative half cycle

       LDB     WSR,#7EH                ; 32 byte window on 1FC0H
       LDB     P1REG_2,#0FFH           ; Turn off pull down
       LDB     P1IO_2,#0FEH            ; P1.0 is output
       LDB     P1SSEL_2,#01H           ; P1.0 is EPA0

       LDB     WSR,#7BH                ; 32 byte window on 1F60H
       LD      EPA_CONTROL0_2,#0FE70H  ; TIMER1, toggle output
       LD      EPA_TIME0_2,#100        ; Initial toggle time

       LDB     WSR,#7CH                ; 32 byte window on 1F80H
       LD      TIMER1_CONTROL_2,#0C2H  ; Enable timer, 1us period

       LD      PTS_SELECT,#0010H       ; Enable PTS interrupt for EPA0
       LDB     INT_MASK,#10H           ; Enable EPA0 interrupts

       EPTS
       EI

SELF:
       SJMP    SELF                    ; Let EPA take over

EPA0_ISR:
       ORB     INT_PEND,#10H           ; Select PTS interrupt
       RET

END
```

270968-6

6.4 Example # 4

This example does not use the PTS, it simply uses the capabilities of the EPA. EPA0 and 1 are remapped together on EPA1. EPA0 sets the pin and resets TIMER1 while EPA1 takes care of resetting the pin. No interrupts are required, however TIMER1 is dedicated to this process alone as it resets in phase with the frequency of the signal produced. This is the fastest PWM frequency possible.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

STK     EQU    200H

CSEG AT 2080H
     DI
     DPTS

     LD      SP,#STK                 ; Initialize stack
     CLRB    INT_PEND                ; Clear out interrupts
     CLRB    INT_PEND1               ;

     LDB     WSR,#7EH                ; 32 byte window on 1FC0H
     LDB     P1REG_2,#0FFH           ; Turn off pull down
     LDB     P1IO_2,#0FDH            ; P1.1 is output
     LDB     P1SSEL_2,#02H           ; P1.1 is EPA1

     LDB     WSR,#7BH                ; 32 byte window on 1F60H
     LD      EPA_CONTROL0_2,#0FE69H  ; TIMER1,set pin,reset timer,lock time
     LD      EPA_TIME0_2,#500        ; Time tag for setting pin
     LD      EPA_CONTROL1_2,#0FF58H  ; TIMER1,reset pin,lock time entry
     LD      EPA_TIME1_2,#150        ; Time to reset pin

     LDB     WSR,#7CH                ; 32 byte window on 1F80H
     LD      TIMER1_CONTROL_2,#0C2H  ; Enable timer, count up, 1us period

SELF:
     SJMP    SELF                    ; Let EPA take over

END
```

270968-7

6.5 Example #5

Finally, the PTS PWM Toggle Mode is used with two channels mapped together to produce a square wave of arbitrary frequency and arbitrary duty cycle. EPA0 is set up to set the pin while EPA1 is set up to reset the pin. Each time each channel performs its function it also performs a PTS interrupt that adds the constant value from its PTS Control Block to the value in its EPA__TIMEn register. The advantage of the example over the previous one is that the timer can be free running and used for other functions as well rather than resetting for every period of the square wave. Note that the constant values of the two PTS channels must be the same to preserve the duty cycle. The duty cycle is chosen by the initial value of the two EPA__TIMEn registers.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

STK     EQU    200H

CSEG AT 2048H
     DCW     PTS_UNUSED0

CSEG AT 2046H
     DCW     PTS_UNUSED1

CSEG AT 2008H
     DCW     EPA0_ISR

CSEG AT 2006H
     DCW     EPA1_ISR

RSEG AT 70H
     PTS_UNUSED0:     DSB   1
     PTS_CONTROL0:    DSB   1
     PTS_SRC0:        DSW   1
     CONST0:          DSW   1

RSEG AT 80H
     PTS_UNUSED1:     DSB   1
     PTS_CONTROL1:    DSB   1
     PTS_SRC1:        DSW   1
     CONST1:          DSW   1

CSEG AT 2080H
     DI
     DPTS

     LD      SP,#STK                ; Initialize stack
     CLRB    INT_PEND               ; Clear out interrupts
     CLRB    INT_PEND1              ;

     LDB     PTS_CONTROL0,#40H
     LD      PTS_SRC0,#EPA_TIME0
     LD      CONST0,#500

     LDB     PTS_CONTROL1,#40H
     LD      PTS_SRC1,#EPA_TIME1
     LD      CONST1,#500
```

270968-8

6.5 Example # 5 (Continued)

```
      LDB     WSR,#7EH                ; 32 byte window on 1FC0H
      LDB     P1REG_2,#0FFH           ; Turn off pull down
      LDB     P1IO_2,#0FDH            ; P1.1 is output
      LDB     P1SSEL_2,#02H           ; P1.1 is EPA1

      LDB     WSR,#7BH                ; 32 byte window on 1F60H
      LD      EPA_CONTROL0_2,#0FE60H  ; TIMER1, set pin
      LD      EPA_TIME0_2,#500        ; 500us high time
      LD      EPA_CONTROL1_2,#0FF50H  ; TIMER1, reset pin
      LD      EPA_TIME1_2,#800        ; 800us low time

      LDB     WSR,#7CH                ; 32 byte window on 1F80H
      LD      TIMER1_CONTROL_2,#0C2H  ; Enable timer, count up, 1us period

      LD      PTS_SELECT,#0018H
      LDB     INT_MASK,#18H

      EPTS
      EI

SELF:
      SJMP    SELF                    ; Let EPA take over

EPA0_ISR:
      ORB     INT_PEND,#10H
      RET

EPA1_ISR:
      ORB     INT_PEND,#08H
      RET

END
```

270968-9

7.0 CODED EXAMPLES

7.1 Using the HSIO to Capture Every Eighth Rising Edge

This program sets up HSI.2 for capturing on every eighth edge. On interrupt, HSI__TIME is read to allow further interrupts to occur.

```
EXAMPLE    MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE (KC.INC)
$LIST

     STK     EQU    200H

CSEG AT 2004H
     DCW     HSI2_ISR

CSEG AT 2080H

     DI
     LD      SP,#STK
     CLRB    INT_PEND
     CLRB    INT_PEND1

     LDB     IOC0,#10H
     LDB     IOC1,#00H
     LDB     IOC2,#80H
     LDB     HSI_MODE,#00H
     LDB     INT_MASK,#04H
     EI

SELF:
     SJMP    SELF

HSI2_ISR:
     PUSHF
     LD      R0,#HSI_TIME
     POPF
     RET

END
```

270968-10

7.2 Using the EPA to Capture Every Nth Edge

This program demonstrates the use of the EPA and PTS to perform the eighth rising edge capture capability of the HSIO. In fact any arbitrary number of rising edges, falling edges, or both can be chosen by adjusting the EPA capture mode and the number of PTS cycles executed before action is taken. This program works by capturing all rising edges on EPA2. 7 rising edges are captured, each creating a PTS interrupt that simply copies EPA__TIME2 to the Zero Register to allow further interrupts. The eighth edge creates a normal interrupt that also reads EPA__TIME2 and resets further interrupts to be PTS interrupts. This algorithm is very flexible because the EPA capture mode can be changed to falling edges or rising and falling edges. Also by adjusting the number of PTS__COUNT2 up to every 255th edge can be captured for interrupt processing.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

STK             EQU 200H
PTS_CYCLES      EQU 8

CSEG AT 2044H
     DCW        PTS_COUNT2

CSEG AT 2004H
     DCW        EPA2_ISR

RSEG AT 70H
     PTS_COUNT2:     DSB  1
     PTS_CONTROL2:   DSB  1
     PTS_SRC2:       DSW  1
     PTS_DEST2:      DSW  1

RSEG AT 76H
     INCR:           DSW  1

CSEG AT 2080H
     DI
     DPTS

     LD      SP,#STK
     CLRB    INT_PEND
     CLRB    INT_PEND1

     LDB     PTS_COUNT2,#PTS_CYCLES  ; 8 PTS cycles then normal interrupt
     LDB     PTS_CONTROL2,#90H       ; PTS Single Transfer Mode
     LD      PTS_SRC2,#EPA_TIME2     ; Source is EPA_TIME2
     LD      PTS_DEST2,#R0           ; Destination is R0

     LDB     WSR,#7EH                ; 32 byte window on 1FC0H
     LDB     P1REG_2,#0FFH           ; Turn off pull down
     LDB     P1IO_2,#0FFH            ; EPA2 is input
     LDB     P1SSEL_2,#04H           ; EPA2 is special function

     LDB     WSR,#7BH                ; 32 byte window on 1F60H
     LD      EPA_CONTROL2_2,#0FE20H  ; TIMER1, capture positive edges
```

270968-11

7.2 Using the EPA to Capture Every Nth Edge (Continued)

```
      LDB     WSR,#7CH                ; 32 byte window on 1F80H
      LD      TIMER1_CONTROL_2,#0C2H  ; Enable TIMER1, 1us period

      LD      PTS_SELECT,#0004H       ; Select PTS interrupt for EPA2
      LDB     INT_MASK,#04H           ; Enable EPA2 interrupts

      EPTS
      EI

SELF:
      SJMP    SELF                    ; Let EPA take over

EPA2_ISR:
      PUSHA
      JBS     PTS_SELECT,1,ERROR      ; Check for bug
      LD      R0,#EPA_TIME2           ; Read EPA_TIME3 to allow new interrupts
      LDB     PTS_COUNT2,#PTS_CYCLES  ; Reset PTS for next 8 rising edges
      OR      PTS_SELECT,#0004H       ; Return to PTS cycles
      POPA
      RET
ERROR:
      ORB     INT_PEND,#04H
      POPA
      RET

END
```

270968-12

7.3 Using the HSIO for an Eight Entry FIFO Unit

The HSIO is set up to perform the functionality of a FIFO in hardware as part of its normal operation. One use of this structure is to capture up to eight events on one pin at a time for interrupt processing, as in this example.

```
EXAMPLE    MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE (KC.INC)
$LIST

     STK     EQU    200H

CSEG AT 2004H
     DCW     HSI2_ISR

CSEG AT 2080H

     DI
     LD      SP,#STK                 ; Initialize Stack Pointer
     CLRB    INT_PEND                ; Clear all interrupts
     CLRB    INT_PEND1               ;

     LDB     IOC0,#40H               ; Enable HSI.3
     LDB     IOC1,#80H               ; HSI interrupt on FIFO full
     LDB     IOC2,#80H               ; Clear entire CAM
     LDB     HSI_MODE,#40H           ; HSI.3 captures rising edges
     LDB     INT_MASK,#04H           ; Enable HSI Data available interrupt
     EI

SELF:
     SJMP    SELF

HSI2_ISR:
     PUSHF
     ORB     IOS1_SAVE,IOS1          ; Clear FIFO by reading out all
     JBC     IOS1_SAVE,7,FIFO_EMPTY  ; entries
     ANDB    IOS1_SAVE,#7FH
     LD      R0,#HSI_TIME
     SJMP    HSI2_ISR
FIFO_EMPTY:
     ANDB    IOS1_SAVE,#7FH
     POPF
     RET

END
```

270968-13

7.4 Using the EPA for an Eight Entry FIFO Unit

This program demonstrates the use of the EPA and PTS to perform the eight entry FIFO function of the HSIO. In fact any arbitrary number of buffered events can be captured up to 256, the maximum number of PTS cycles without interrupt. The events occurring on one channel are stored separately from the events on another channel thus removing the need for decode to discover where the event originated. The software FIFO is created by the PTS cycles where the EPA__TIME3 value is copied to memory and then the destination address is incremented. After the desired number of events have been captured, eight here, a normal interrupt occurs allowing processing of the information just obtained. This example only utilizes one channel, EPA3, and therefore only creates one software FIFO.

```
EXAMPLE     MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

STK             EQU 200H
PTS_CYCLES      EQU 8

CSEG AT 2042H
     DCW        PTS_COUNT3

CSEG AT 2002H
     DCW        EPA3_ISR

RSEG AT 70H
     PTS_COUNT3:     DSB  1
     PTS_CONTROL3:   DSB  1
     PTS_SRC3:       DSW  1
     PTS_DEST3:      DSW  1

RSEG AT 76H
     CLRREG:         DSW  1

RSEG AT 80H
     FIFO:           DSW  8

CSEG AT 2080H
     DI
     DPTS

     LD      SP,#STK                ; Initialize Stack Pointer
     CLRB    INT_PEND               ; Clear all pending interrupts
     CLRB    INT_PEND1              ;

     LD      CLRREG,#FIFO

CLEAR:
     ST      R0,[CLRREG]+           ; Routine to clear register area for FIFO
     CMP     CLRREG,#00A0H
     BNE     CLEAR
```

270968-14

7.4 Using the EPA for an Eight Entry FIFO Unit (Continued)

```
      LDB     PTS_COUNT3,#PTS_CYCLES  ; 8 PTS cycles then normal interrupt
      LDB     PTS_CONTROL3,#85H       ; Single xsfer, incr and update dest
      LD      PTS_SRC3,#EPA_TIME3     ; Source is EPA_TIME3
      LD      PTS_DEST3,#FIFO         ; Destination is FIFO area

      LDB     WSR,#7EH                ; 32 byte window on 1FC0H
      LDB     P1REG_2,#0FFH           ; Turn off pull down
      LDB     P1IO_2,#0FFH            ; P1.3 is input
      LDB     P1SSEL_2,#08H           ; P1.3 is EPA3

      LDB     WSR,#7BH                ; 32 byte window on 1F60H
      LD      EPA_CONTROL3_2,#0FE20H  ; TIMER1, capture positive edges

      LDB     WSR,#7CH                ; 32 byte window on 1F80H
      LD      TIMER1_CONTROL_2,#0C6H  ; Enable TIMER1, 16us period

      LD      PTS_SELECT,#0002H       ; Select PTS interrupt for EPA3
      ORB     INT_MASK,#02H           ; Enable EPA3 interrupts

      EPTS
      EI

SELF:
      SJMP    SELF                    ; Let EPA take over

EPA3_ISR:
      PUSHA
      JBS     PTS_SELECT,1,ERROR      ; Check for bug
      LDB     PTS_COUNT3,#PTS_CYCLES  ; Reset PTS for next 8 rising edges
      LDB     PTS_DEST3,#FIFO         ; Put dest back at beginning of FIFO
      OR      PTS_SELECT,#0002H       ; Return to PTS cycles
      POPA
      RET
ERROR:
      ORB     INT_PEND,#02H
      POPA
      RET

END
```

270968-15

7.5 Using the EPA to Perform Multiple Output Events in Succession

This program demonstrates the use of the EPA and PTS to perform the eight entry CAM of the HSIO for the scheduling of multiple events on one output channel. In fact any arbitrary number of buffered events can be set up to occur limited only by the speed of the PTS to set up the next pending event and the maximum number of PTS cycles without software intervention (256).

```
EXAMPLE     MODULE MODE, STACKSIZE(20)

$NOLIST
$INCLUDE(KR.INC)
$LIST

PTS_CYCLES          EQU 8

CSEG AT 2042H
       DCW          PTS_COUNT3

CSEG AT 2002H
       DCW          EPA3_ISR

RSEG AT 070H
     PTS_COUNT3:     DSB  1
     PTS_CONTROL3:   DSB  1
     PTS_SRC3:       DSW  1
     PTS_DEST3:      DSW  1

RSEG AT 076H
     TEMPREG:        DSW  1

RSEG AT 080H
     CAM0:           DSW  1
     CAM1:           DSW  1
     CAM2:           DSW  1
     CAM3:           DSW  1
     CAM4:           DSW  1
     CAM5:           DSW  1
     CAM6:           DSW  1

CSEG AT 2080H

     DI
     DPTS

     LD      TEMPREG,#CAM0

CLEAR:                                  ; Clearing register area for CAM
     ST      R0,[TEMPREG]+
     CMP     TEMPREG,#00A0H
     BNE     CLEAR
     LD      TEMPREG,#CAM0
```

270968-16

7.5 Using the EPA to Perform Multiple Output Events in Succession (Continued)

```
LOAD:                                   ; Loading "CAM" for many timed events
      LD       CAM0,#007DH              ;
      LD       CAM1,#00C8H              ;
      LD       CAM2,#00D1H              ;
      LD       CAM3,#0113H              ;
      LD       CAM4,#0145H              ;
      LD       CAM5,#0168H              ;
      LD       CAM6,#0172H              ;

      LDB      PTS_COUNT3,#PTS_CYCLES   ; 8 PTS cycles then normal interrupt
      LDB      PTS_CONTROL3,#8AH        ; Single transfer mode,incr and update src
      LD       PTS_SRC3,#CAM0           ; Source is CAM
      LD       PTS_DEST3,#EPA_TIME3     ; Destination is FIFO area

      LDB      WSR,#7EH                 ; 32 byte window on 1FC0H
      LDB      P1REG_2,#0F7H            ; P1.3 begins low
      LDB      P1IO_2,#0F7H             ; P1.3 is output
      LDB      P1SSEL_2,#08H            ; P1.3 is EPA3

      LDB      WSR,#7BH                 ; 32 byte window on 1F60H
      LD       EPA_CONTROL3_2,#0FE70H   ; TIMER1, toggle output
      LD       EPA_TIME3_2,#0032H       ; Load EPA_TIME3

      LDB      WSR,#7CH                 ; 32 byte window on 1F80H
      LD       COMP_CONTROL0_2,#0FE49H  ; TIMER1, lock time entry, reset timer
      LD       COMP_TIME0_2,#0172H      ; Reset timer after 370 counts

      LD       TIMER1_2,#0FFC0H         ; Set TIMER1 to count up to 0000H
      LD       TIMER1_CONTROL_2,#0C6H   ; Enable TIMER1, 16us period

      LD       PTS_SELECT,#0002H        ; Select PTS interrupt for EPA3
      ORB      INT_MASK,#02H            ; Enable EPA3 interrupts

      EPTS
      EI

SELF:
      SJMP     SELF                     ; Let EPA take over

EPA3_ISR:
      PUSHA
      JBS      PTS_SELECT,1,ERROR       ; Check for bug
      LDB      PTS_COUNT3,#PTS_CYCLES   ; Reset PTS for next 8 rising edges
      LDB      PTS_SRC3,#CAM0           ; Put dest back at beginning of FIFO
      LDB      WSR,#7BH                 ; 32 byte window on 1F60H
      LD       EPA_TIME3_2,#0032H       ; Reset first event time
      OR       PTS_SELECT,#0002H        ; Return to PTS cycles
      POPA
      RET
ERROR:
      ORB      INT_PEND,#02H            ; Send to PTS cycle
      POPA
      RET

END
```

270968-17

7.6 Using EPA2 to Clock TIMER2 Internally

The KR device has some interesting characteristics due to the fact that the T2CLK and T2DIR pins are shared with EPA pins. Because of the way the output drivers are set up it is possible to configure an EPA channel as an output and drive one of these two clock pins with no additional hardware. This example uses EPA0 to drive T2CLK and hence clock TIMER2 "externally". Note that one can then base an EPA output event on TIMER2. Interestingly, if TIMER1 counts at its slowest rate, 16 μs period, the TIMER2 is clocked at its slowest rate, once every 64 edges, it is possible to schedule an event to occur approximately 50 days in the future.

```
EXAMPLE       MODULE MAIN, STACKSIZE(20)

$NOLIST
$INCLUDE (KR.INC)
$LIST

STK     EQU    200H

CSEG AT 2080H

      LD      SP,#STK
      CLRB    INT_PEND
      CLRB    INT_PEND1

      LDB     WSR,#1FH                ; 128 byte window on 1F80H
      LDB     P1REG_0,#0FEH           ; Turn off pull down on P1.0
      LDB     P1IO_0,#0FEH            ; P1.0 is output
      LDB     P1SSEL_0,#01H           ; P1.0 is T2CLK

      LDB     WSR,#1EH                ; 128 byte window on 1F00H
      LD      EPA_CONTROL3_0,#0FE78H  ; TIMER1, toggle output, lock time
      LD      EPA_TIME3_0,#500        ; Time tag for toggle

      LDB     WSR,#3EH
      LDB     TIMER1_CONTROL_1,#0C6H  ; Enable timer, count up, 16us period
      LDB     TIMER2_CONTROL_1,#0CEH  ; Enable timer, external clock

SELF:
      SJMP    SELF                    ; Let EPA take over

END
```

270968-18

APPENDIX A

```
;*******************************************************************************
; SFRS.KR - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS
;               OF THE 8XC196KR
;               (C) INTEL CORPORATION 1989
;*******************************************************************************
R0              EQU  00H:WORD    ; R    ZERO REGISTER
ZERO            EQU  00H:WORD    ; R    ZERO REGISTER
PTS_SELECT      EQU  04H:WORD    ; R/W
PTS_SRV         EQU  06H:WORD    ; R/W
INT_MASK        EQU  08H:BYTE    ; R/W
INT_PEND        EQU  09H:BYTE    ; R/W
WATCHDOG        EQU  0AH:BYTE    ;   W  WATCHDOG TIMER
INT_PEND1       EQU  12H:BYTE    ; R/W
INT_MASK1       EQU  13H:BYTE    ; R/W
WSR             EQU  14H:BYTE    ; R/W
SP              EQU  18H:WORD    ; R/W

;*******************************************************************************
; SFR DEFINITIONS FOR REGISTERS OUTSIDE 00 TO FFH ALONG WITH ALL
; WINDOW DEFINITIONS FOR THE SAME REGISTER.
;*******************************************************************************

;LABEL:                                      FOR USE WITH WSR VALUE:
;---------------                             ---------------------------

P0PIN             EQU   01FDAH:BYTE ; R/W    ---
P0PIN_0           EQU     0DAH:BYTE ; R/W    1FH
P0PIN_1           EQU     0DAH:BYTE ; R/W    3FH
P0PIN_2           EQU     0FAH:BYTE ; R/W    7EH

P1PIN             EQU   01FD6H:BYTE ; R      ---
P1PIN_0           EQU     0D6H:BYTE ; R      1FH
P1PIN_1           EQU     0D6H:BYTE ; R      3FH
P1PIN_2           EQU     0F6H:BYTE ; R      7EH
P1REG             EQU   01FD4H:BYTE ; R/W    ---
P1REG_0           EQU     0D4H:BYTE ; R/W    1FH
P1REG_1           EQU     0D4H:BYTE ; R/W    3FH
P1REG_2           EQU     0F4H:BYTE ; R/W    7EH
P1IO              EQU   01FD2H:BYTE ; R/W    ---
P1IO_0            EQU     0D2H:BYTE ; R/W    1FH
P1IO_1            EQU     0D2H:BYTE ; R/W    3FH
P1IO_2            EQU     0F2H:BYTE ; R/W    7EH
P1SSEL            EQU   01FD0H:BYTE ; R/W    ---
P1SSEL_0          EQU     0D0H:BYTE ; R/W    1FH
P1SSEL_1          EQU     0D0H:BYTE ; R/W    3FH
P1SSEL_2          EQU     0F0H:BYTE ; R/W    7EH

P2PIN             EQU   01FCFH:BYTE ; R      ---
P2PIN_0           EQU     0CFH:BYTE ; R      1FH
P2PIN_1           EQU     0CFH:BYTE ; R      3FH
P2PIN_2           EQU     0EFH:BYTE ; R      7EH
P2REG             EQU   01FCDH:BYTE ; R/W    ---
P2REG_0           EQU     0CDH:BYTE ; R/W    1FH
P2REG_1           EQU     0CDH:BYTE ; R/W    3FH
P2REG_2           EQU     0EDH:BYTE ; R/W    7EH
P2IO              EQU   01FCBH:BYTE ; R/W    ---
```

270968-19

```
P2IO_0          EQU     0CBH:BYTE ; R/W    1FH
P2IO_1          EQU     0CBH:BYTE ; R/W    3FH
P2IO_2          EQU     0EBH:BYTE ; R/W    7EH
P2SSEL          EQU   01FC9H:BYTE ; R/W    ---
P2SSEL_0        EQU     0C9H:BYTE ; R/W    1FH
P2SSEL_1        EQU     0C9H:BYTE ; R/W    3FH
P2SSEL_2        EQU     0E9H:BYTE ; R/W    7EH

P3PIN           EQU   01FFEH:BYTE ; R      ---
P3PIN_0         EQU     0FEH:BYTE ; R      1FH
P3PIN_1         EQU     0FEH:BYTE ; R      3FH
P3PIN_2         EQU     0FEH:BYTE ; R      7FH
P3REG           EQU   01FFCH:BYTE ; R/W    ---
P3REG_0         EQU     0FCH:BYTE ; R/W    1FH
P3REG_1         EQU     0FCH:BYTE ; R/W    3FH
P3REG_2         EQU     0FCH:BYTE ; R/W    7FH

P4PIN           EQU   01FFFH:BYTE ; R      ---
P4PIN_0         EQU     0FFH:BYTE ; R      1FH
P4PIN_1         EQU     0FFH:BYTE ; R      3FH
P4PIN_2         EQU     0FFH:BYTE ; R      7FH
P4REG           EQU   01FFDH:BYTE ; R/W    ---
P4REG_0         EQU     0FDH:BYTE ; R/W    1FH
P4REG_1         EQU     0FDH:BYTE ; R/W    3FH
P4REG_2         EQU     0FDH:BYTE ; R/W    7FH

P5PIN           EQU   01FF7H:BYTE ; R      ---
P5PIN_0         EQU     0F7H:BYTE ; R      1FH
P5PIN_1         EQU     0F7H:BYTE ; R      3FH
P5PIN_2         EQU     0F7H:BYTE ; R      7FH
P5REG           EQU   01FF5H:BYTE ; R/W    ---
P5REG_0         EQU     0F5H:BYTE ; R/W    1FH
P5REG_1         EQU     0F5H:BYTE ; R/W    3FH
P5REG_2         EQU     0F5H:BYTE ; R/W    7FH
P5IO            EQU   01FF3H:BYTE ; R/W    ---
P5IO_0          EQU     0F3H:BYTE ; R/W    1FH
P5IO_1          EQU     0F3H:BYTE ; R/W    3FH
P5IO_2          EQU     0F3H:BYTE ; R/W    7FH
P5SSEL          EQU   01FF1H:BYTE ; R/W    ---
P5SSEL_0        EQU     0F1H:BYTE ; R/W    1FH
P5SSEL_1        EQU     0F1H:BYTE ; R/W    3FH
P5SSEL_2        EQU     0F1H:BYTE ; R/W    7FH

P6PIN           EQU   01FD7H:BYTE ; R      ---
P6PIN_0         EQU     0D7H:BYTE ; R      1FH
P6PIN_1         EQU     0D7H:BYTE ; R      3FH
P6PIN_2         EQU     0F7H:BYTE ; R      7EH
P6REG           EQU   01FD5H:BYTE ; R/W    ---
P6REG_0         EQU     0D5H:BYTE ; R/W    1FH
P6REG_1         EQU     0D5H:BYTE ; R/W    3FH
P6REG_2         EQU     0F5H:BYTE ; R/W    7EH
P6IO            EQU   01FD3H:BYTE ; R/W    ---
P6IO_0          EQU     0D3H:BYTE ; R/W    1FH
P6IO_1          EQU     0D3H:BYTE ; R/W    3FH
P6IO_2          EQU     0F3H:BYTE ; R/W    7EH
P6SSEL          EQU   01FD1H:BYTE ; R/W    ---
P6SSEL_0        EQU     0D1H:BYTE ; R/W    1FH
P6SSEL_1        EQU     0D1H:BYTE ; R/W    3FH
```

270968–20

```
P6SSEL_2            EQU     0F1H:BYTE ; R/W    7EH

TIMER1              EQU   01F9AH:WORD ; R/W    ---
TIMER1_0            EQU     09AH:WORD ; R/W    1FH
TIMER1_1            EQU     0DAH:WORD ; R/W    3EH
TIMER1_2            EQU     0FAH:WORD ; R/W    7CH
TIMER1_CONTROL      EQU   01F98H:BYTE ; R/W    ---
TIMER1_CONTROL_0    EQU     098H:BYTE ; R/W    1FH
TIMER1_CONTROL_1    EQU     0D8H:BYTE ; R/W    3EH
TIMER1_CONTROL_2    EQU     0F8H:BYTE ; R/W    7CH
TIMER2              EQU   01F9EH:WORD ; R/W    ---
TIMER2_0            EQU     09EH:WORD ; R/W    1FH
TIMER2_1            EQU     0DEH:WORD ; R/W    3EH
TIMER2_2            EQU     0FEH:WORD ; R/W    7CH
TIMER2_CONTROL      EQU   01F9CH:BYTE ; R/W    ---
TIMER2_CONTROL_0    EQU     09CH:BYTE ; R/W    1FH
TIMER2_CONTROL_1    EQU     0DCH:BYTE ; R/W    3EH
TIMER2_CONTROL_2    EQU     0FCH:BYTE ; R/W    7CH

SP_BAUD             EQU   01FBCH:WORD ;   W    ---
SP_BAUD_0           EQU     0BCH:WORD ;   W    1FH
SP_BAUD_1           EQU     0FCH:WORD ;   W    3EH
SP_BAUD_2           EQU     0FCH:WORD ;   W    7DH
SP_CONTROL          EQU   01FBBH:BYTE ; R/W    ---
SP_CONTROL_0        EQU     0BBH:BYTE ; R/W    1FH
SP_CONTROL_1        EQU     0FBH:BYTE ; R/W    3EH
SP_CONTROL_2        EQU     0FBH:BYTE ; R/W    7DH
SP_STATUS           EQU   01FB9H:BYTE ; R/W    ---
SP_STATUS_0         EQU     0B9H:BYTE ; R/W    1FH
SP_STATUS_1         EQU     0F9H:BYTE ; R/W    3EH
SP_STATUS_2         EQU     0F9H:BYTE ; R/W    7DH
SBUF_TX             EQU   01FBAH:BYTE ; R/W    ---
SBUF_TX_0           EQU     0BAH:BYTE ; R/W    1FH
SBUF_TX_1           EQU     0FAH:BYTE ; R/W    3EH
SBUF_TX_2           EQU     0FAH:BYTE ; R/W    7DH
SBUF_RX             EQU   01FB8H:BYTE ; R/W    ---
SBUF_RX_0           EQU     0B8H:BYTE ; R/W    1FH
SBUF_RX_1           EQU     0F8H:BYTE ; R/W    3EH
SBUF_RX_2           EQU     0F8H:BYTE ; R/W    7DH

EPAIPV              EQU   01FA8H:BYTE ; R      ---
EPAIPV_0            EQU     0A8H:BYTE ; R      1FH
EPAIPV_1            EQU     0E8H:BYTE ; R      3EH
EPAIPV_2            EQU     0E8H:BYTE ; R      7DH
EPA_PEND            EQU   01FA2H:WORD ; R/W    ---
EPA_PEND_0          EQU     0A2H:WORD ; R/W    1FH
EPA_PEND_1          EQU     0E2H:WORD ; R/W    3EH
EPA_PEND_2          EQU     0E2H:WORD ; R/W    7DH
EPA_PEND1           EQU   01FA6H:BYTE ; R/W    ---
EPA_PEND1_0         EQU     0A6H:BYTE ; R/W    1FH
EPA_PEND1_1         EQU     0E6H:BYTE ; R/W    3EH
EPA_PEND1_2         EQU     0E6H:BYTE ; R/W    7DH
EPA_MASK            EQU   01FA0H:WORD ; R/W    ---
EPA_MASK_0          EQU     0A0H:WORD ; R/W    1FH
EPA_MASK_1          EQU     0E0H:WORD ; R/W    3EH
EPA_MASK_2          EQU     0E0H:WORD ; R/W    7DH
EPA_MASK1           EQU   01FA4H:WORD ; R/W    ---  BUG...must write as word
EPA_MASK1_0         EQU     0A4H:WORD ; R/W    1FH
```

270968-21

```
EPA_MASK1_1        EQU     0E4H:WORD ; R/W    3EH
EPA_MASK1_2        EQU     0E4H:WORD ; R/W    7DH

USFR               EQU   01FF6H:BYTE ;   W    ---
USFR_0             EQU     0F6H:BYTE ;   W    1FH
USFR_1             EQU     0F6H:BYTE ;   W    3FH
USFR_2             EQU     0F6H:BYTE ;   W    7FH

SLPCMD             EQU   01FFAH:BYTE ; R/W    ---
SLPCMD_0           EQU     0FAH:BYTE ; R/W    1FH
SLPCMD_1           EQU     0FAH:BYTE ; R/W    3FH
SLPCMD_2           EQU     0FAH:BYTE ; R/W    7FH
SLPSTAT            EQU   01FF8H:BYTE ; R/W    ---
SLPSTAT_0          EQU     0F8H:BYTE ; R/W    1FH
SLPSTAT_1          EQU     0F8H:BYTE ; R/W    3FH
SLPSTAT_2          EQU     0F8H:BYTE ; R/W    7FH
SLPFUNREG          EQU   01FFBH:BYTE ; R/W    ---
SLPFUNREG_0        EQU     0FBH:BYTE ; R/W    1FH
SLPFUNREG_1        EQU     0FBH:BYTE ; R/W    3FH
SLPFUNREG_2        EQU     0FBH:BYTE ; R/W    7FH

AD_TIME            EQU   01FAFH:BYTE ; R/W    ---
AD_TIME_0          EQU     0AFH:BYTE ; R/W    1FH
AD_TIME_1          EQU     0EFH:BYTE ; R/W    3EH
AD_TIME_2          EQU     0EFH:BYTE ; R/W    7DH
AD_TEST            EQU   01FAEH:BYTE ; R/W    ---
AD_TEST_0          EQU     0AEH:BYTE ; R/W    1FH
AD_TEST_1          EQU     0EEH:BYTE ; R/W    3EH
AD_TEST_2          EQU     0EEH:BYTE ; R/W    7DH
AD_COMMAND         EQU   01FACH:BYTE ; R/W    ---
AD_COMMAND_0       EQU     0ACH:BYTE ; R/W    1FH
AD_COMMAND_1       EQU     0ECH:BYTE ; R/W    3EH
AD_COMMAND_2       EQU     0ECH:BYTE ; R/W    7DH
AD_RESULT          EQU   01FAAH:WORD ; R/W    ---
AD_RESULT_0        EQU     0AAH:WORD ; R/W    1FH
AD_RESULT_1        EQU     0EAH:WORD ; R/W    3EH
AD_RESULT_2        EQU     0EAH:WORD ; R/W    7DH

SSIO_BAUD          EQU   01FB4H:BYTE ;   W    ---
SSIO_BAUD_0        EQU     0B4H:BYTE ;   W    1FH
SSIO_BAUD_1        EQU     0F4H:BYTE ;   W    3EH
SSIO_BAUD_2        EQU     0F4H:BYTE ;   W    7DH
SSIO_STCR1         EQU   01FB3H:BYTE ; R/W    ---
SSIO_STCR1_0       EQU     0B3H:BYTE ; R/W    1FH
SSIO_STCR1_1       EQU     0F3H:BYTE ; R/W    3EH
SSIO_STCR1_2       EQU     0F3H:BYTE ; R/W    7DH
SSIO_STB1          EQU   01FB2H:BYTE ; R/W    ---
SSIO_STB1_0        EQU     0B2H:BYTE ; R/W    1FH
SSIO_STB1_1        EQU     0F2H:BYTE ; R/W    3EH
SSIO_STB1_2        EQU     0F2H:BYTE ; R/W    7DH
SSIO_STCR0         EQU   01FB1H:BYTE ; R/W    ---
SSIO_STCR0_0       EQU     0B1H:BYTE ; R/W    1FH
SSIO_STCR0_1       EQU     0F1H:BYTE ; R/W    3EH
SSIO_STCR0_2       EQU     0F1H:BYTE ; R/W    7DH
SSIO_STB0          EQU   01FB0H:BYTE ; R/W    ---
SSIO_STB0_0        EQU     0B0H:BYTE ; R/W    1FH
SSIO_STB0_1        EQU     0F0H:BYTE ; R/W    3EH
SSIO_STB0_2        EQU     0F0H:BYTE ; R/W    7DH
```

270968-22

```
COMP_TIME1          EQU   01F8EH:WORD ; R/W    ---
COMP_TIME1_0        EQU     08EH:WORD ; R/W    1FH
COMP_TIME1_1        EQU     0CEH:WORD ; R/W    3EH
COMP_TIME1_2        EQU     0EEH:WORD ; R/W    7CH
COMP_CONTROL1       EQU   01F8CH:WORD ; R/W    ---
COMP_CONTROL1_0     EQU     08CH:WORD ; R/W    1FH
COMP_CONTROL1_1     EQU     0CCH:WORD ; R/W    3EH
COMP_CONTROL1_2     EQU     0ECH:WORD ; R/W    7CH
COMP_TIME0          EQU   01F8AH:WORD ; R/W    ---
COMP_TIME0_0        EQU     08AH:WORD ; R/W    1FH
COMP_TIME0_1        EQU     0CAH:WORD ; R/W    3EH
COMP_TIME0_2        EQU     0EAH:WORD ; R/W    7CH
COMP_CONTROL0       EQU   01F88H:WORD ; R/W    ---
COMP_CONTROL0_0     EQU     088H:WORD ; R/W    1FH
COMP_CONTROL0_1     EQU     0C8H:WORD ; R/W    3EH
COMP_CONTROL0_2     EQU     0E8H:WORD ; R/W    7CH

EPA_TIME9           EQU   01F86H:WORD ; R/W    ---
EPA_TIME9_0         EQU     086H:WORD ; R/W    1FH
EPA_TIME9_1         EQU     0C6H:WORD ; R/W    3EH
EPA_TIME9_2         EQU     0E6H:WORD ; R/W    7CH
EPA_CONTROL9        EQU   01F84H:WORD ; R/W    ---
EPA_CONTROL9_0      EQU     084H:WORD ; R/W    1FH
EPA_CONTROL9_1      EQU     0C4H:WORD ; R/W    3EH
EPA_CONTROL9_2      EQU     0E4H:WORD ; R/W    7CH
EPA_TIME8           EQU   01F82H:WORD ; R/W    ---
EPA_TIME8_0         EQU     082H:WORD ; R/W    1FH
EPA_TIME8_1         EQU     0C2H:WORD ; R/W    3EH
EPA_TIME8_2         EQU     0E2H:WORD ; R/W    7CH
EPA_CONTROL8        EQU   01F80H:WORD ; R/W    ---
EPA_CONTROL8_0      EQU     080H:WORD ; R/W    1FH
EPA_CONTROL8_1      EQU     0C0H:WORD ; R/W    3EH
EPA_CONTROL8_2      EQU     0E0H:WORD ; R/W    7CH
EPA_TIME7           EQU   01F7EH:WORD ; R/W    ---
EPA_TIME7_0         EQU     0FEH:WORD ; R/W    1EH
EPA_TIME7_1         EQU     0FEH:WORD ; R/W    3DH
EPA_TIME7_2         EQU     0FEH:WORD ; R/W    7BH
EPA_CONTROL7        EQU   01F7CH:WORD ; R/W    ---
EPA_CONTROL7_0      EQU     0FCH:WORD ; R/W    1EH
EPA_CONTROL7_1      EQU     0FCH:WORD ; R/W    3DH
EPA_CONTROL7_2      EQU     0FCH:WORD ; R/W    7BH
EPA_TIME6           EQU   01F7AH:WORD ; R/W    ---
EPA_TIME6_0         EQU     0FAH:WORD ; R/W    1EH
EPA_TIME6_1         EQU     0FAH:WORD ; R/W    3DH
EPA_TIME6_2         EQU     0FAH:WORD ; R/W    7BH
EPA_CONTROL6        EQU   01F78H:WORD ; R/W    ---
EPA_CONTROL6_0      EQU     0F8H:WORD ; R/W    1EH
EPA_CONTROL6_1      EQU     0F8H:WORD ; R/W    3DH
EPA_CONTROL6_2      EQU     0F8H:WORD ; R/W    7BH
EPA_TIME5           EQU   01F76H:WORD ; R/W    ---
EPA_TIME5_0         EQU     0F6H:WORD ; R/W    1EH
EPA_TIME5_1         EQU     0F6H:WORD ; R/W    3DH
EPA_TIME5_2         EQU     0F6H:WORD ; R/W    7BH
EPA_CONTROL5        EQU   01F74H:WORD ; R/W    ---
EPA_CONTROL5_0      EQU     0F4H:WORD ; R/W    1EH
EPA_CONTROL5_1      EQU     0F4H:WORD ; R/W    3DH
EPA_CONTROL5_2      EQU     0F4H:WORD ; R/W    7BH
```

270968-23

```
EPA_TIME4          EQU   01F72H:WORD ; R/W    ---
EPA_TIME4_0        EQU     0F2H:WORD ; R/W    1EH
EPA_TIME4_1        EQU     0F2H:WORD ; R/W    3DH
EPA_TIME4_2        EQU     0F2H:WORD ; R/W    7BH
EPA_CONTROL4       EQU   01F70H:WORD ; R/W    ---
EPA_CONTROL4_0     EQU     0F0H:WORD ; R/W    1EH
EPA_CONTROL4_1     EQU     0F0H:WORD ; R/W    3DH
EPA_CONTROL4_2     EQU     0F0H:WORD ; R/W    7BH
EPA_TIME3          EQU   01F6EH:WORD ; R/W    ---
EPA_TIME3_0        EQU     0EEH:WORD ; R/W    1EH
EPA_TIME3_1        EQU     0EEH:WORD ; R/W    3DH
EPA_TIME3_2        EQU     0EEH:WORD ; R/W    7BH
EPA_CONTROL3       EQU   01F6CH:WORD ; R/W    ---
EPA_CONTROL3_0     EQU     0ECH:WORD ; R/W    1EH
EPA_CONTROL3_1     EQU     0ECH:WORD ; R/W    3DH
EPA_CONTROL3_2     EQU     0ECH:WORD ; R/W    7BH
EPA_TIME2          EQU   01F6AH:WORD ; R/W    ---
EPA_TIME2_0        EQU     0EAH:WORD ; R/W    1EH
EPA_TIME2_1        EQU     0EAH:WORD ; R/W    3DH
EPA_TIME2_2        EQU     0EAH:WORD ; R/W    7BH
EPA_CONTROL2       EQU   01F68H:WORD ; R/W    ---
EPA_CONTROL2_0     EQU     0E8H:WORD ; R/W    1EH
EPA_CONTROL2_1     EQU     0E8H:WORD ; R/W    3DH
EPA_CONTROL2_2     EQU     0E8H:WORD ; R/W    7BH
EPA_TIME1          EQU   01F66H:WORD ; R/W    ---
EPA_TIME1_0        EQU     0E6H:WORD ; R/W    1EH
EPA_TIME1_1        EQU     0E6H:WORD ; R/W    3DH
EPA_TIME1_2        EQU     0E6H:WORD ; R/W    7BH
EPA_CONTROL1       EQU   01F64H:WORD ; R/W    ---
EPA_CONTROL1_0     EQU     0E4H:WORD ; R/W    1EH
EPA_CONTROL1_1     EQU     0E4H:WORD ; R/W    3DH
EPA_CONTROL1_2     EQU     0E4H:WORD ; R/W    7BH
EPA_TIME0          EQU   01F62H:WORD ; R/W    ---
EPA_TIME0_0        EQU     0E2H:WORD ; R/W    1EH
EPA_TIME0_1        EQU     0E2H:WORD ; R/W    3DH
EPA_TIME0_2        EQU     0E2H:WORD ; R/W    7BH
EPA_CONTROL0       EQU   01F60H:WORD ; R/W    ---
EPA_CONTROL0_0     EQU     0E0H:WORD ; R/W    1EH
EPA_CONTROL0_1     EQU     0E0H:WORD ; R/W    3DH
EPA_CONTROL0_2     EQU     0E0H:WORD ; R/W    7BH
```

270968-24