

DESIGNER'S NOTEBOOK



TMS320C5x Interrupt Response Time

Contributed by Jeff Beinart

Design Problem

What are the important issues in TMS320C5x interrupt latency/processing?

Solution

This design note calculates the speed at which the TMS320C5x can recognize consecutive interrupts. This time depends on the interrupt latency and the time required to service the interrupt. A practical application, a bar code scanner routine, will be used as an example.

Figure 1 shows a bar code where the bar widths and spacings between bars are different, signifying a specific number associated with each bar. To determine the width of a scanned bar, external hardware must generate an interrupt at both the leading and trailing edges of the bar. The TMS320C5x context switches to an ISR, which copies the contents of a Timer Register out to data memory. At some later time we can subtract the timer values, corresponding to each interrupt, and multiply by the Timer clock period to find the bar width.

The Timer clock frequency is a key parameter. A higher frequency yields a more accurate calculation.

$$\text{Bar Code Width} = (\text{Difference in Timer Values}) * (\text{CLKOUT1 period})$$

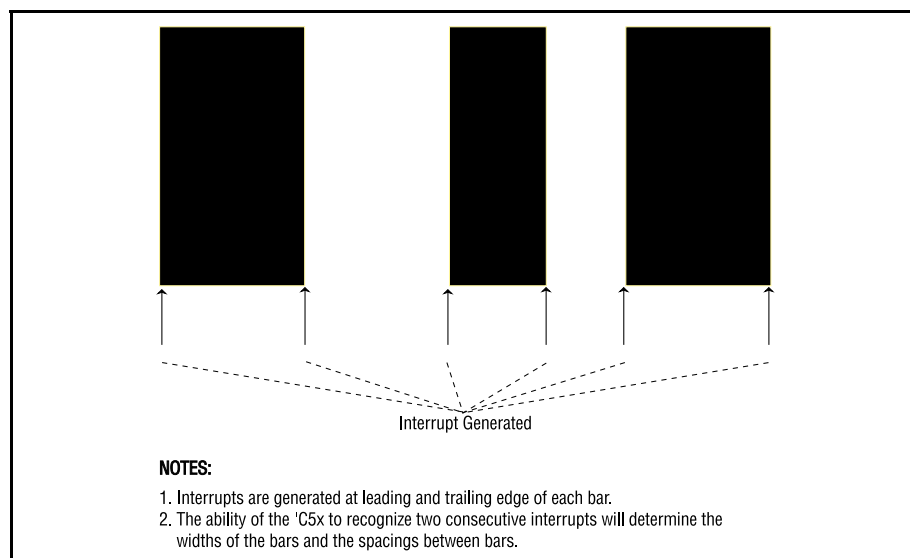


Figure 1. Example of a bar code with interrupts generated at edges

Problem Definition

1. Use a TMS320C5x-80 running at an internal machine rate of 40 MHz (internal machine period of 25 ns per cycle).
2. There are two stages to the Timer with each stage consisting of a period register and a counter register. Only the 16-bit wide TIM counter and PRD period registers of the second stage of the timer will be used. This means the ISR will copy the TIM counter register to data memory.
3. In this application, the ISR consists of four instructions, thereby requiring a total of four locations in the vector table, thus reducing the external interrupt capability of the 'C5x by one.
4. The interrupt latency of the 'C5x depends on the current contents of the pipeline. The device always completes all instructions in the pipeline before executing the soft vector. It is up to the software engineer not to use multiple-cycle instructions or non-interruptible instructions (such as RPT & RPTZ) during the inner loop routine of the bar code scanner routine.
5. The inner loop of the bar code scanner routine is small enough to be run out of internal program space. This could be from internal ROM or from internal program RAM. This will minimize the execution time of copying the Timer Counter register to data memory.
6. The ISR dedicates AR5 as a pointer to the memory location where the TIM counter register gets written. AR5 is incremented within the ISR so that the information is constantly stored in different memory locations. AR5 cannot be used in the main program.

ISR Implementation

An external interrupt is generated whenever the edge of a bar code is detected. The 'C5x will recognize the interrupt and vector to an ISR which copies the TIM Timer counter register out to data memory and returns to the main routine. The ISR consists of the following instructions:

```
MAR  *,AR5 ; Load ARP with 5
LAMB TIM   ; Load lower half of ACC with TIM counter
SACL **    ; Store ACC low indirectly to memory
RETE      ; Return to main program and enable interrupts
```

TIM is the Timer counter register, which is a peripheral memory-mapped register. The LAMB is a single-word, two-cycle instruction. All the other instructions are single-word, single-cycle instructions.

The RETE is specified as a four-cycle instruction because the pipeline is flushed on the return from the ISR. It therefore takes four cycles to execute the next instruction in the main routine. Since as soon as the RETE is executed interrupts can occur, RETE can be considered a single-cycle instruction.

Figure 2 illustrates what the pipeline looks like when two consecutive interrupts are recognized by the 'C5x. The first interrupt occurs before cycle 1, whereas the second interrupt occurs before cycle 13. The significant pipeline events are summarized after Figure 2.

| | | | | | | | | | |
|---|------|------|------|------|------|------|------|------|------|
| 1st interrupt | | | | | | | | | |
| Cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Fetch | I1 | I2 | I3 | I4 | I5 | I6 | D | D | D |
| Decode | – | I1 | I2 | I3 | I4 | I5 | INTR | D | D |
| Read | – | – | I1 | I2 | I3 | I4 | I5 | INTR | D |
| Execute | – | – | – | I1 | I2 | I3 | I4 | I5 | INTR |
| 2nd interrupt occurs in this period | | | | | | | | | |
| Cycle | 9 | 10 | 11 | 12 | 13 | 14 | (15) | 16 | 17 |
| Fetch | MAR | LAMM | SACL | D | RETE | D | D | D | I6 |
| Decode | D | MAR | LAMM | SACL | D | RETE | D | D | – |
| Read | D | D | MAR | LAMM | D | SACL | RETE | D | – |
| Execute | D | D | D | MAR | LAMM | LAMM | SACL | RETE | – |
| Cycle | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| Fetch | D | D | D | MAR | LAMM | SACL | D | RETE | D |
| Decode | INTR | D | D | D | MAR | LAMM | SACL | D | RETE |
| Read | – | INTR | D | D | D | MAR | LAMM | D | SACL |
| Execute | – | – | INTR | D | D | D | MAR | LAMM | LAMM |
| Cycle | (27) | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| Fetch | D | D | I6 | | | | | | |
| Decode | D | D | – | | | | | | |
| Read | RETE | D | – | | | | | | |
| Execute | SACL | RETE | – | | | | | | |
| Legend I represents the specific instruction number, n D stands for DUMMY cycle MAR Modify Auxiliary Register LAMM Load Accumulator low with contents of TIM register SACL Store Accumulator Low out to Data Memory RETE Return from interrupt and clear INTM bit (globally enable interrupts) | | | | | | | | | |

Figure 2. Calculating the minimum time between interrupts

Explanation of Figure 2

1. Cycle 0 – 1 High-to-low transition of “1st” external interrupt occurs before the fetch of I2.
2. Cycle 3 Interrupt must be held low for three clock cycles, whereupon it is recognized by the CPU.
3. Cycle 4 Appropriate bit of Interrupt Flag Register (IFR) is set signifying an interrupt has occurred.
4. Cycle 5 I6 is fetched, however, it will be refetched after the return from interrupt.
5. Cycle 6 INTR jammed into the pipeline.

-
- | | |
|-----------------|---|
| 6. Cycle 9 | Start of first instruction in the ISR. INTM is set to a 1 (INTM=1) and an \overline{IACK} is generated. INTM is a global interrupt bit and will globally disable any interrupts from occurring. \overline{IACK} clears an internal 'C5x FF and allows a bit in the IFR register to be set by the next external interrupt. |
| 7. Cycle 10 | LAMM, a 1-word, 2-cycle instruction, is fetched. |
| 8. Cycle 9 – 13 | High-to-low transition of “2nd” external interrupt (could actually occur sometime after cycle 9 and before cycle 13). |
| 9. Cycle 15 | SACL instruction executes completing the write of the TIM Timer counter register to data memory [pointer register (AR5) is updated by 1]. |
| 10. Cycle 16 | RETE instruction executes causing a return to the main program and causing INTM=0 (global interrupt enabled). The pending (second) interrupt can be recognized starting at the next cycle. |
| 11. Cycle 17 | I6 is fetched, however, it will be refetched after the return from interrupt. |
| 12. Cycle 18 | INTR jammed into the pipeline. |
| 13. Cycle 27 | SACL instruction executes completing the write of the TIM Timer counter register, corresponding to the second interrupt, to data memory. |

In this analysis, the 'C5x cannot distinguish as to when the actual interrupt occurred between cycles 9 and 13. It will respond to the interrupts in the same way. Let's assume the second interrupt occurs right before cycle 10. It will take four cycles from when the second interrupt occurs until it is recognized by the CPU and the Interrupt Flag Register bit of the IFR is set. The processor will wait until the INTM = 0 before recognizing the new interrupt. This occurs at cycle 16. Although I6 is fetched in cycle 17, it is discarded when INTR is jammed into the pipeline at cycle 18.

The Timer counter register values are copied to data memory in cycles 15 and 27. Therefore the bar code width is calculated as follows:

$$\begin{aligned}
 \text{Bar Code Width} &= (\text{Clock cycle 27} - \text{Clock cycle 15}) * (\text{CLKOUT1 period}) \\
 &= (12 \text{ cycles}) * (25 \text{ ns per cycle}) \\
 &= (12 * 25 \text{ ns}) \\
 &= 300 \text{ ns}
 \end{aligned}$$

In the above calculation it was assumed the second interrupt occurred between cycles 9 and 13. If the second interrupt occurs at cycle 14, then the reading of the Timer value to data memory will be delayed by one additional cycle (if it occurs at cycle 15, then it will be delayed by two cycles and so forth).

Conclusion

The TMS320C5x is able to recognize interrupts every 12 clock cycles. There are three instructions, or four cycles consumed by the ISR. Since the 'C5x is a pipelined architecture, it is difficult to calculate the interrupt latency. For this analysis the best case interrupt latency is found to be eight cycles (12 – 4).