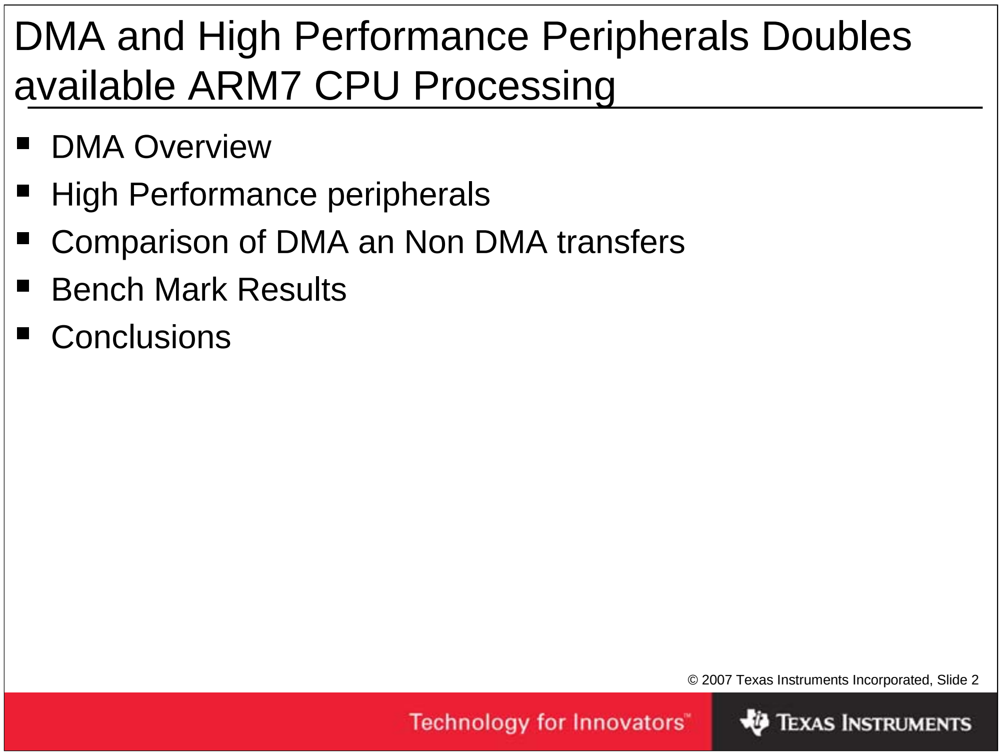
Using DMA with High Performance Peripherals to Maximize System Performance

John Mangino

WW TMS470 Catalog Applications



DMA and High Performance Peripherals Doubles available ARM7 CPU Processing

- DMA Overview
- High Performance peripherals
- Comparison of DMA an Non DMA transfers
- Bench Mark Results
- Conclusions

DMA – Direct memory Access Overview

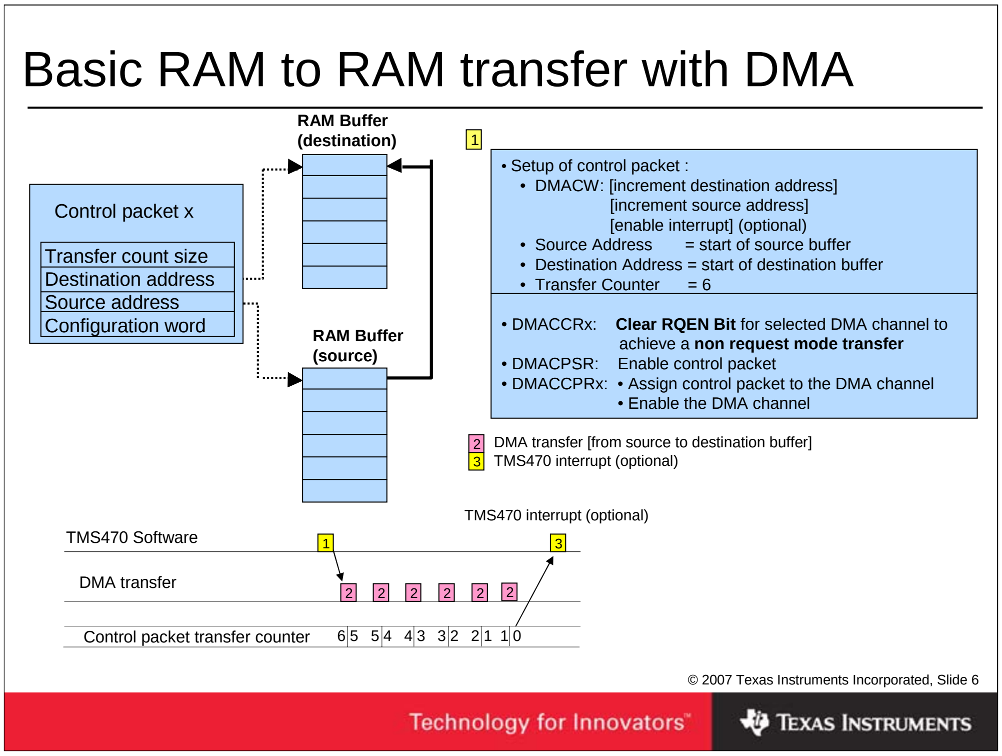
- Direct Memory Access (DMA) transfers data between memory and peripheral locations
- The data transfers take place in parallel with CPU activity, maximizing system performance
- Data can be transferred concurrently with CPU transactions as long as there is no resource conflict (such as may occur when both the CPU and DMA controller attempt to access the same bus)
- Data transfers are not interrupt driven, the system performance is maximized

TMS470 DMA Controller

- TMS470 DMA utilizes 32 control packets and 16 channels
  - Control packets control DMA transfers
  - DMA channels connect peripherals to DMA controller
- Configurable transfer access size is to byte, half-word, or word transfers
- Block transfer size is configurable to allow a set number of DMA transactions (1 to 65,535)
- Control packet allows for interrupt enable after completion of block transfers
- Each channel may be individually enabled

DMA Usage

- Transfer data between memory locations
- Transfer data between memory and peripherals
- Transfer data between peripherals



Basic RAM to RAM transfer with DMA

RAM Buffer (destination)

Control packet x
- Transfer count size
- Destination address
- Source address
- Configuration word

RAM Buffer (source)

1
- Setup of control packet :
  - DMACW: [increment destination address] [increment source address] [enable interrupt] (optional)
  - Source Address = start of source buffer
  - Destination Address = start of destination buffer
  - Transfer Counter = 6

- DMACCRx: **Clear RQEN Bit** for selected DMA channel to achieve a **non request mode transfer**
- DMACPSR: Enable control packet
- DMACCPRx: • Assign control packet to the DMA channel • Enable the DMA channel

2 DMA transfer [from source to destination buffer]
3 TMS470 interrupt (optional)

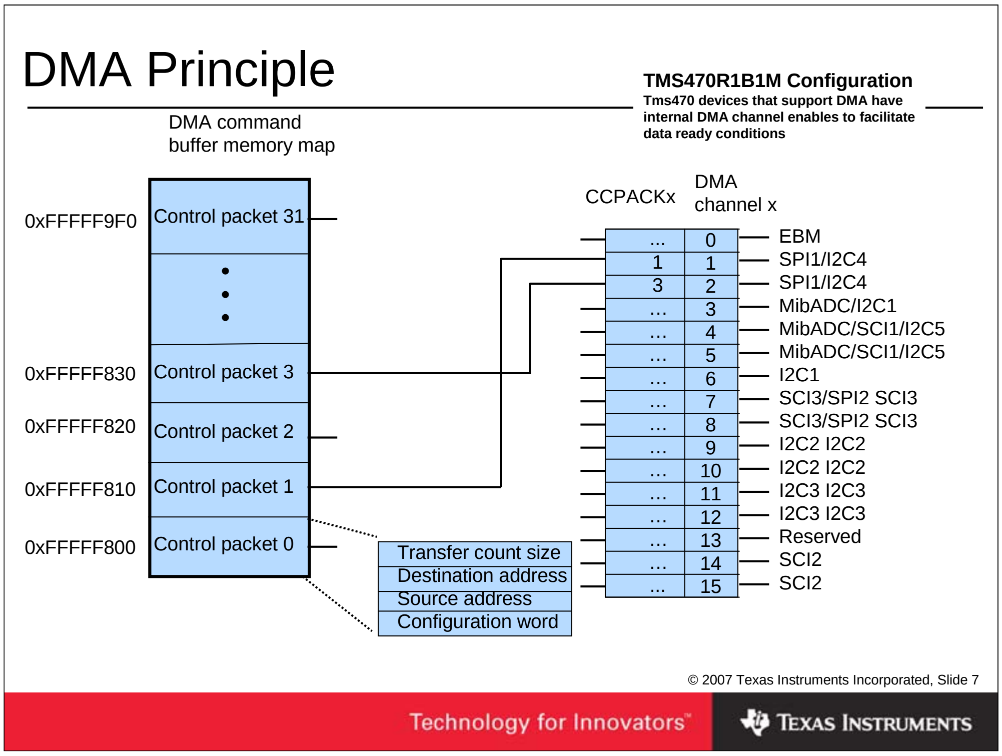
TMS470 interrupt (optional)

TMS470 Software 1 3

DMA transfer 2 2 2 2 2 2

Control packet transfer counter 6|5 5|4 4|3 3|2 2|1 1|0

Classic DMA transfers are between memory locations. The transfers are done with no CPU cycles except for the initialization of the DMA.



DMA Principle

TMS470R1B1M Configuration

Tms470 devices that support DMA have internal DMA channel enables to facilitate data ready conditions

CCPACKx	DMA channel x	
…	0	EBM
1	1	SPI1/I2C4
3	2	SPI1/I2C4
…	3	MibADC/I2C1
…	4	MibADC/SCI1/I2C5
…	5	MibADC/SCI1/I2C5
…	6	I2C1
…	7	SCI3/SPI2 SCI3
…	8	SCI3/SPI2 SCI3
…	9	I2C2 I2C2
…	10	I2C2 I2C2
…	11	I2C3 I2C3
…	12	I2C3 I2C3
…	13	Reserved
…	14	SCI2
…	15	SCI2

The TMS470 devices have DMA channel enables that connect the peripheral to the DMA. The enable line are the data ready for transfer to and from the peripheral. The DMA manages up to 16 channels, and supports data transfer for both on-chip and off-chip memories and peripherals. The DMA controller

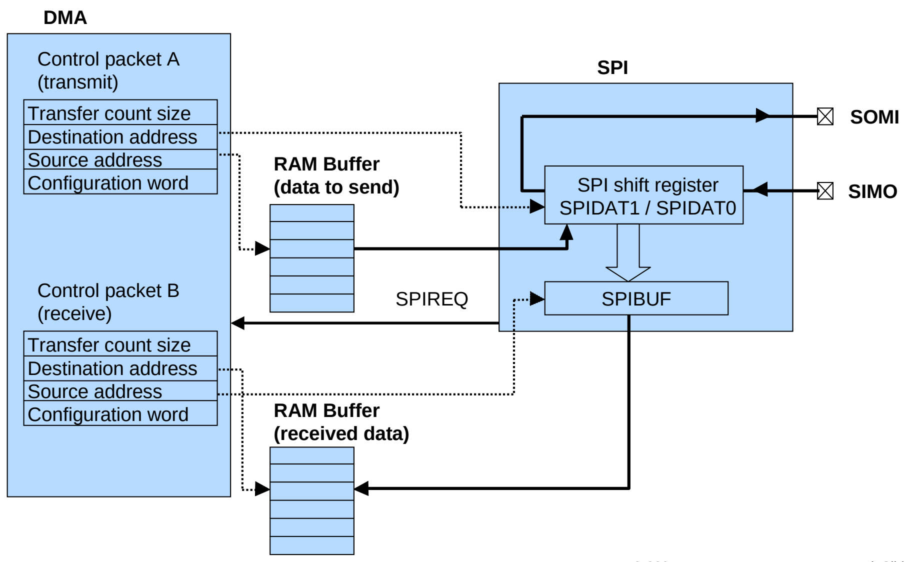
is connected to both the CPU and peripheral buses, enabling these data transfers to occur in parallel with CPU activity and thus maximizing overall system performance. Each channel has two control packets attached to it, allowing the DMA to continuously load RAM and generate periodic interrupts so that the data can be read by the CPU. The control packets allow for the interrupt enable, and the channels determine the priority level of the interrupt.

DMA transfers occur in one of two modes:

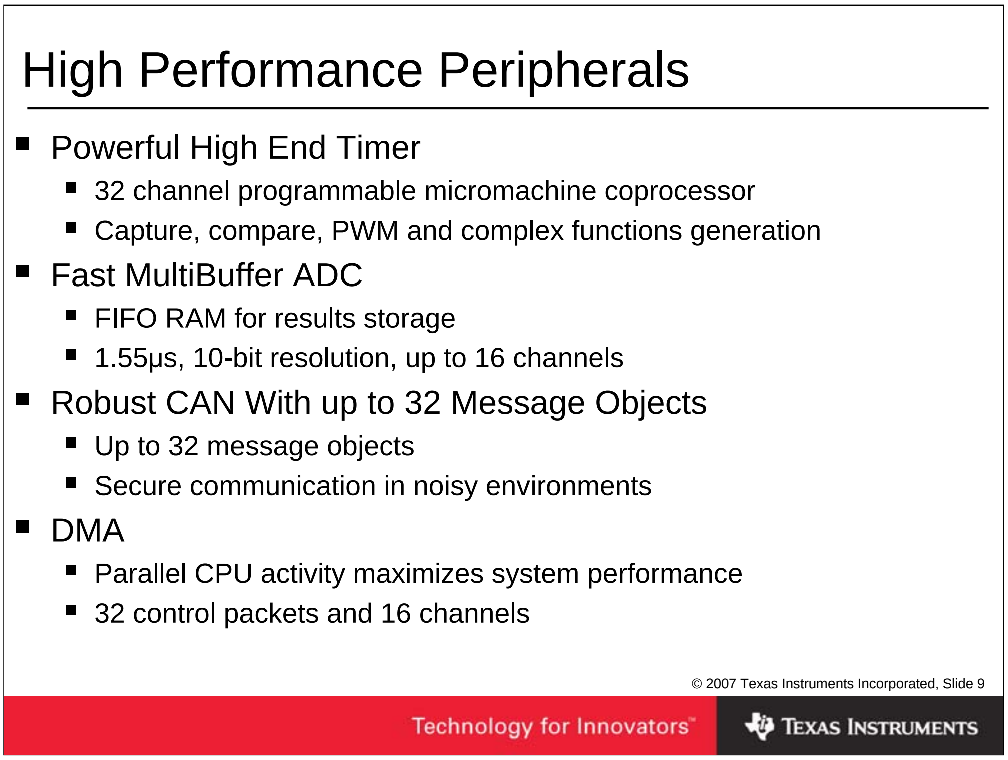
· Non-request mode (used when transferring from memory to memory)

· Request mode (used when transferring from memory to peripheral)

SPI Transmit and Receive Data with DMA



This diagram show the DMA between memory and the SPI peripheral. This setup enable SPI transfers with the need for CPU interrupts.



High Performance Peripherals

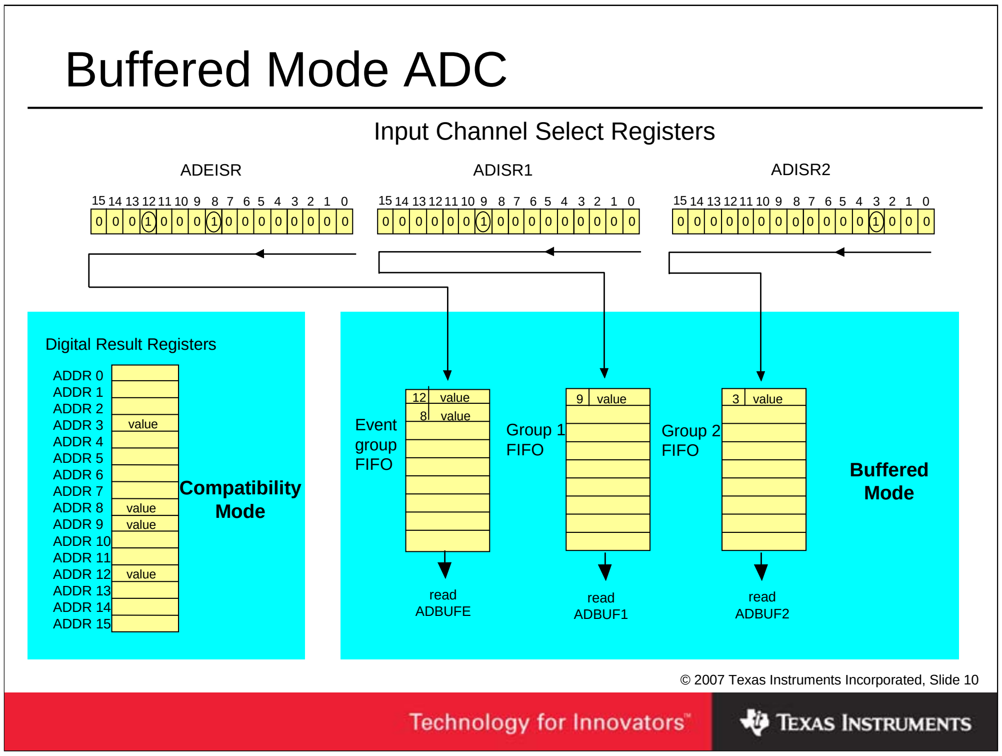
- Powerful High End Timer
  - 32 channel programmable micromachine coprocessor
  - Capture, compare, PWM and complex functions generation
- Fast MultiBuffer ADC
  - FIFO RAM for results storage
  - 1.55μs, 10-bit resolution, up to 16 channels
- Robust CAN With up to 32 Message Objects
  - Up to 32 message objects
  - Secure communication in noisy environments
- DMA
  - Parallel CPU activity maximizes system performance
  - 32 control packets and 16 channels

The MultiBuffer ADC is fast, has lots of channels, and the multibuffered feature off loads the CPU

The High End Timer is a powerful RISC coprocessor providing significant additional system performance and flexibility

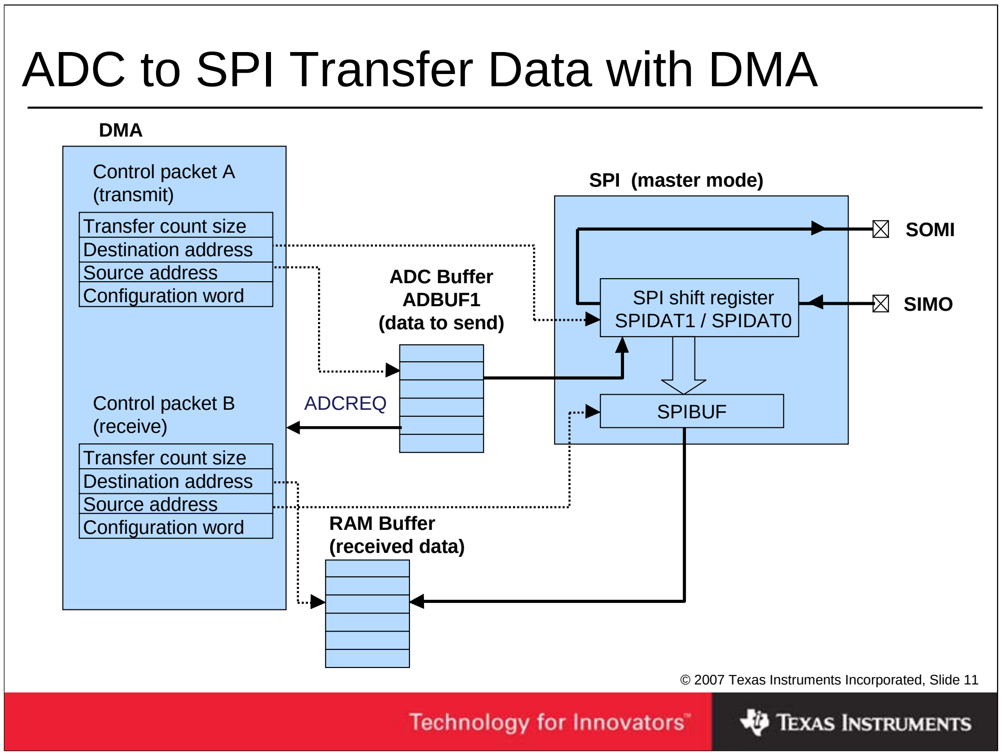
The High End Can Controller provides high level messaging without additional CPU overhead.

The Direct Memory Access (DMA) controller transfers data between address ranges in the memory map without intervention by the CPU, maximizing system performance. When coupled with the other peripherals the DMA can significantly off load the CPU.

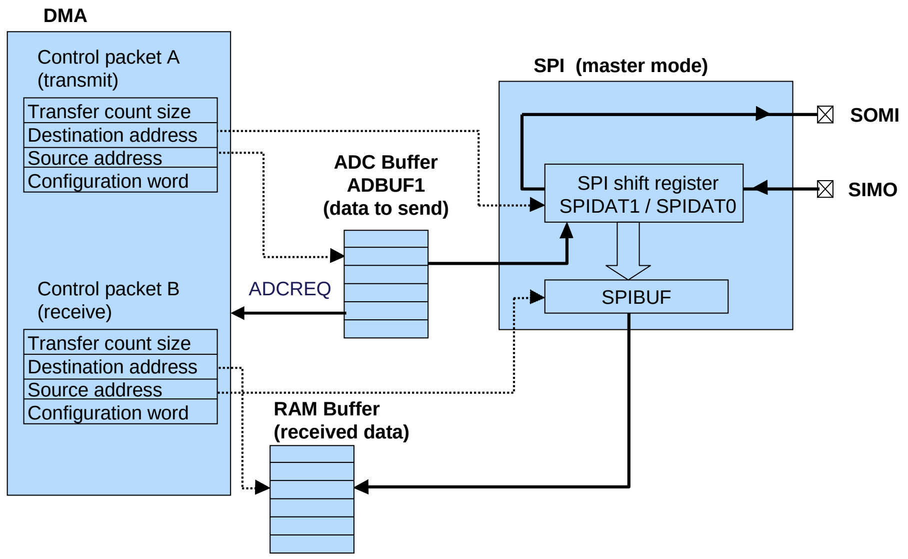


Buffered Mode ADC

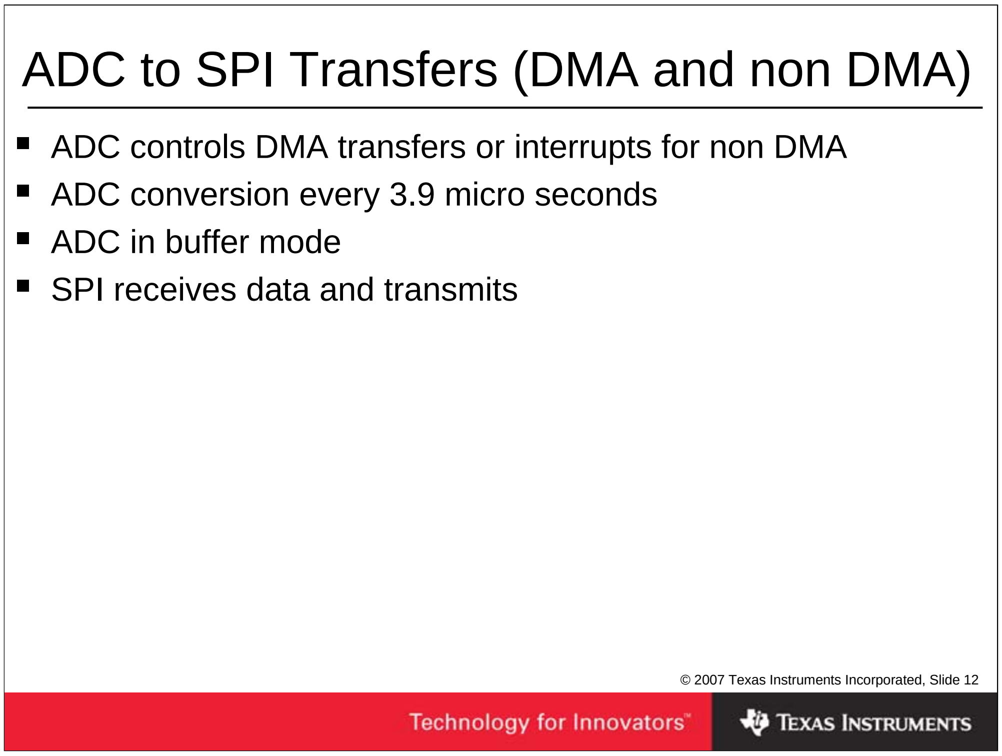
This slide shows very clearly the difference between Compatibility and Buffered Mode. In Compatibility mode, when each group has a particular channel selected, that channel's results are placed in the corresponding Digital Result Register. In Buffered mode each Group's conversions are placed in the corresponding buffer in the order in which they were received.



ADC to SPI Transfer Data with DMA

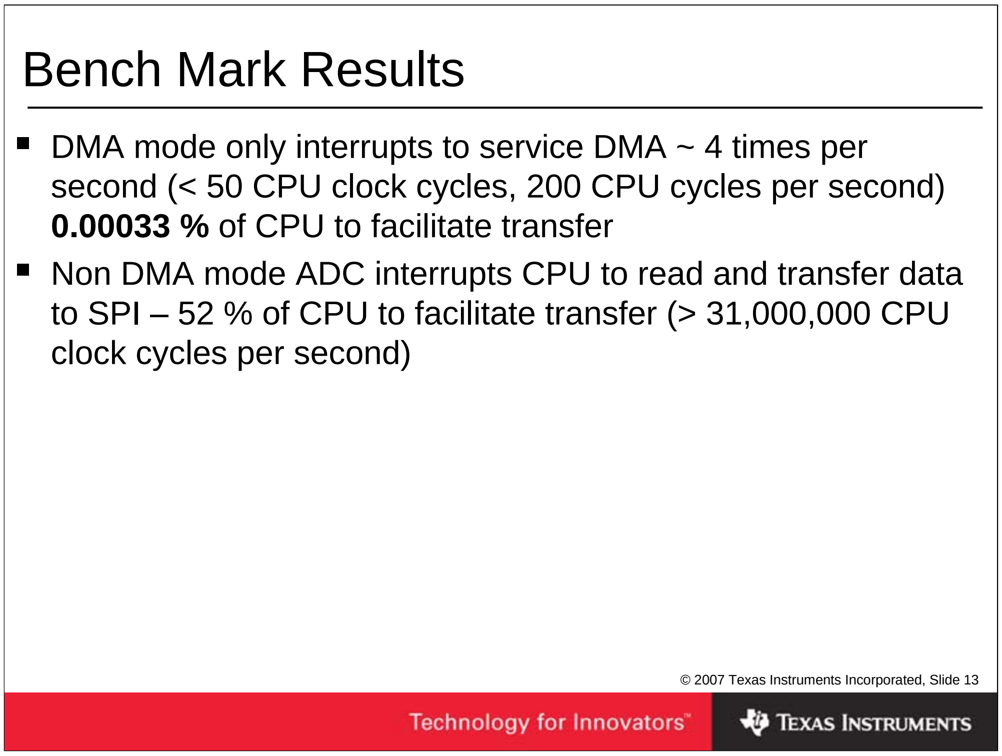


This example show the DMA transferring data between the ADC and the SPI. The ADC is set up in an auto convert mode and the data is transferred to the SPI for transmission to an external device. The following slide will compare the use of DMA verses a non DMA implementaion.



ADC to SPI Transfers (DMA and non DMA)

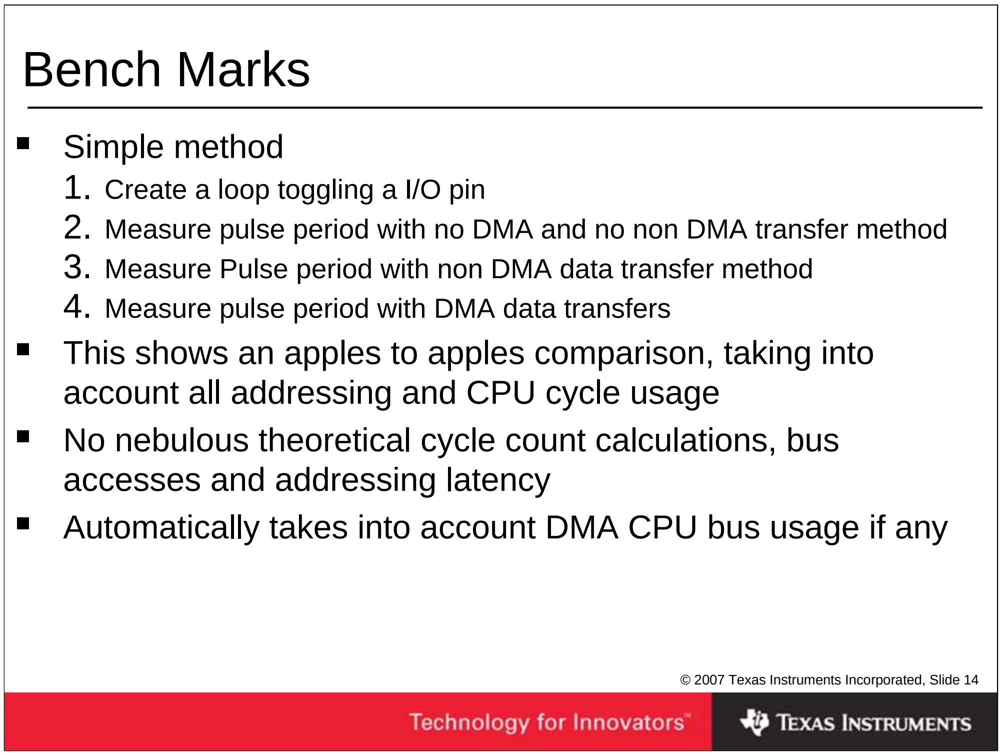
- ADC controls DMA transfers or interrupts for non DMA
- ADC conversion every 3.9 micro seconds
- ADC in buffer mode
- SPI receives data and transmits



Bench Mark Results

- DMA mode only interrupts to service DMA ~ 4 times per second (< 50 CPU clock cycles, 200 CPU cycles per second) **0.00033 %** of CPU to facilitate transfer
- Non DMA mode ADC interrupts CPU to read and transfer data to SPI – 52 % of CPU to facilitate transfer (> 31,000,000 CPU clock cycles per second)

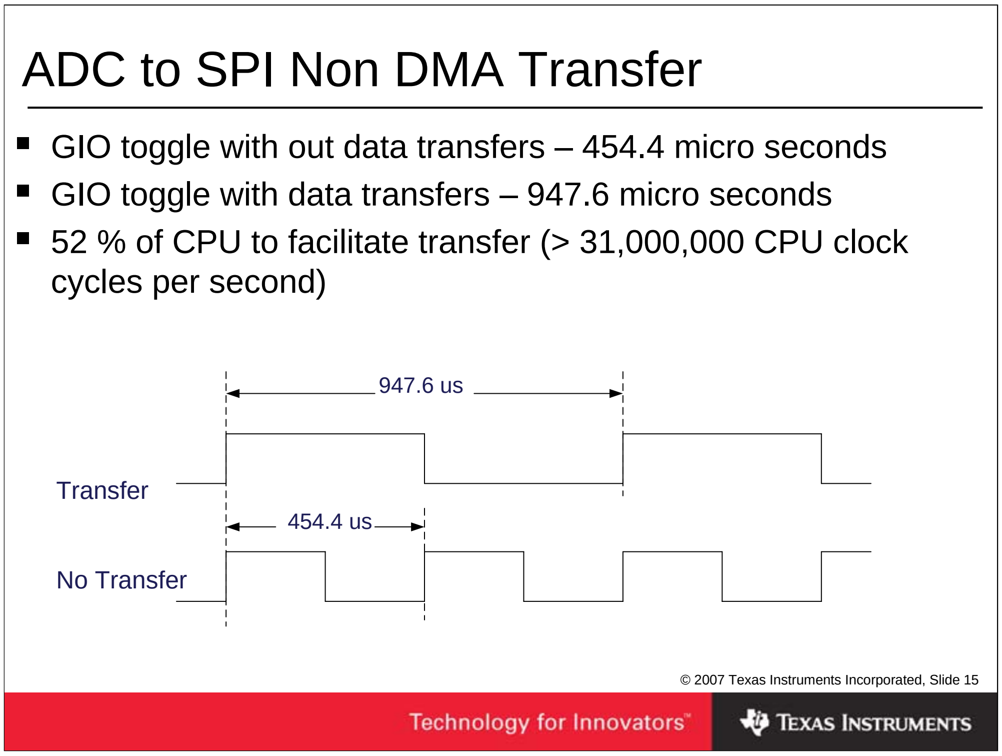
The bench mark comparing the use of DMA verses a interrupt driven code is incredible. The ADC is converting data at 3.9 microseconds or 256,000 samples per second. Interrupt driven code to transfer the data to the SPI takes over 50% of the CPU cycles verses only 200 CPU cycles per second, or 0.00033%.



Bench Marks

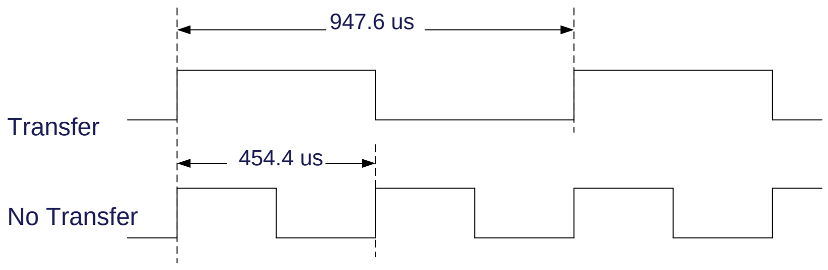
- Simple method
  1. Create a loop toggling a I/O pin
  2. Measure pulse period with no DMA and no non DMA transfer method
  3. Measure Pulse period with non DMA data transfer method
  4. Measure pulse period with DMA data transfers
- This shows an apples to apples comparison, taking into account all addressing and CPU cycle usage
- No nebulous theoretical cycle count calculations, bus accesses and addressing latency
- Automatically takes into account DMA CPU bus usage if any

The test set up uses a IO toggle that compares the period of the toggling IO pin between DMA and the non DMA method.

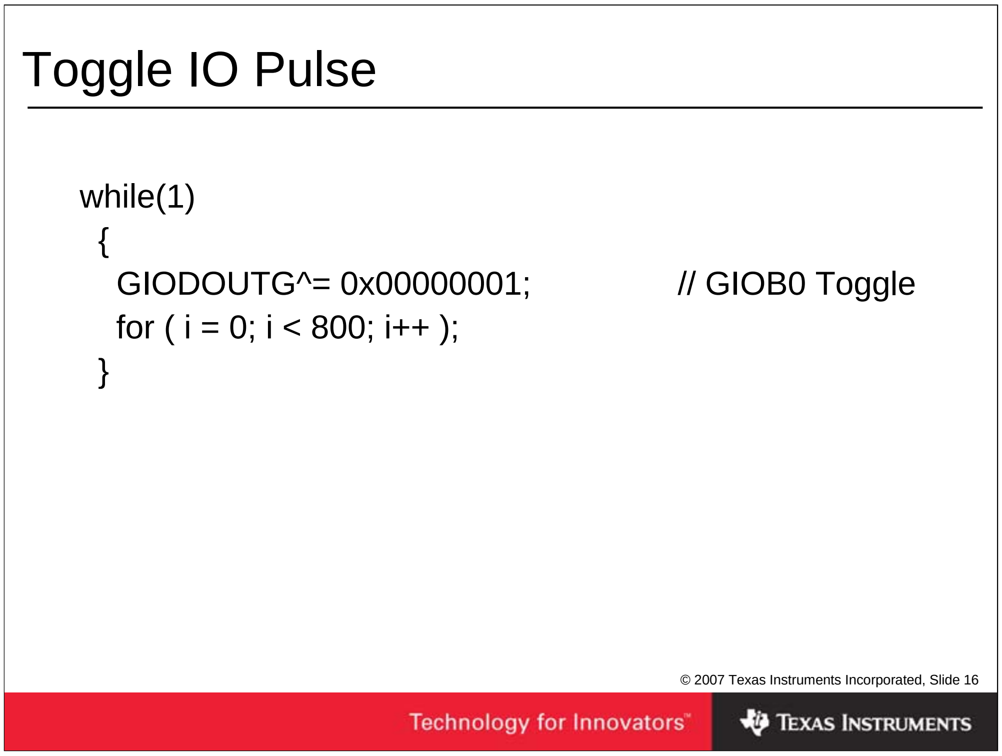


ADC to SPI Non DMA Transfer

- GIO toggle with out data transfers – 454.4 micro seconds
- GIO toggle with data transfers – 947.6 micro seconds
- 52 % of CPU to facilitate transfer (> 31,000,000 CPU clock cycles per second)



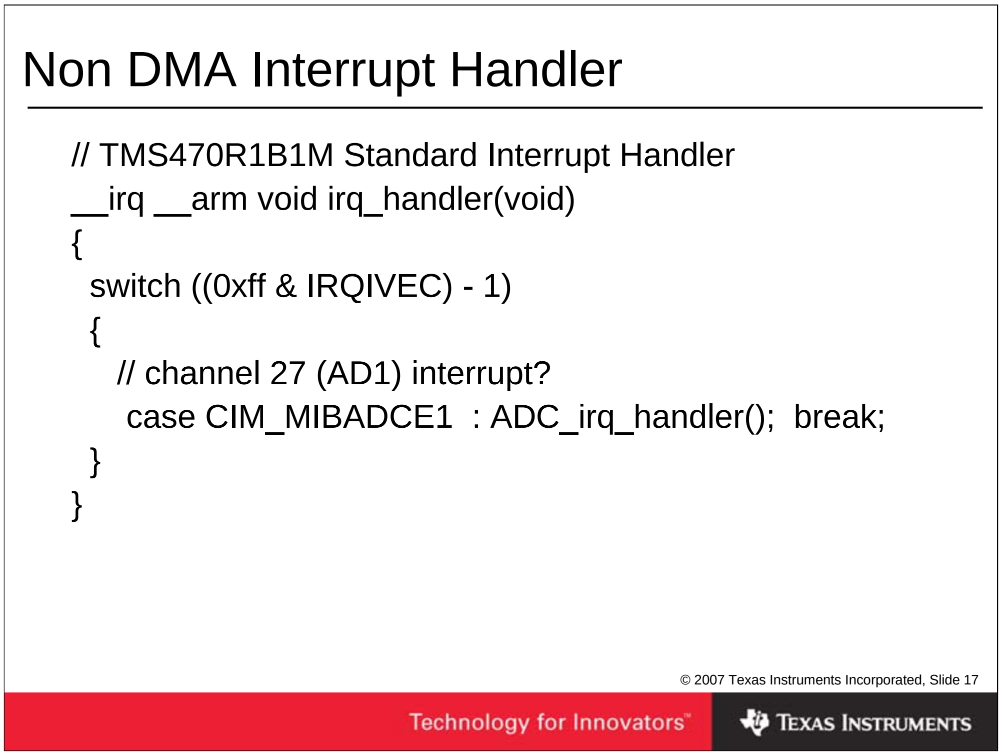
With a CPU clock of 60 MHz the IO toggles with a period of 454.4 micro seconds with no interrupts for data transfer. When the interrupts are enabled the IO toggle period is lengthened to 947.6 microseconds. This shows the need CPU cycle to service the interrupts verses no data transfers.



Toggle IO Pulse

```
while(1)
  {
    GIODOUTG^= 0x00000001;                // GIOB0 Toggle
    for ( i = 0; i < 800; i++ );
  }
```

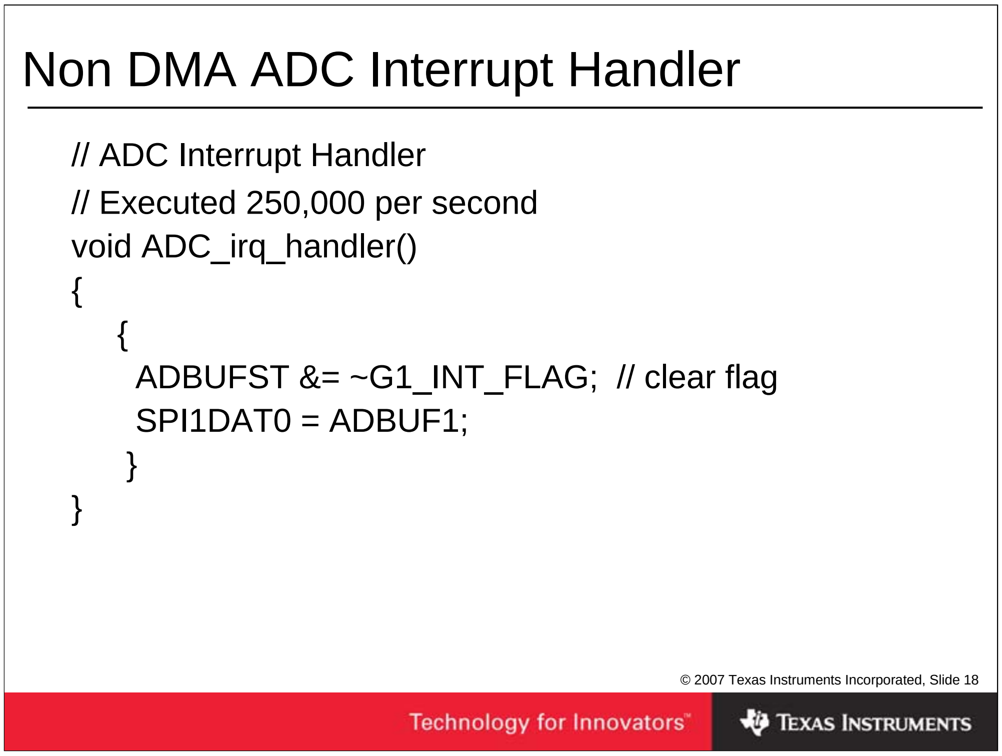
This the simple toggle IO routine.



Non DMA Interrupt Handler

```
// TMS470R1B1M Standard Interrupt Handler
__irq __arm void irq_handler(void)
{
  switch ((0xff & IRQIVEC) - 1)
  {
     // channel 27 (AD1) interrupt?
      case CIM_MIBADCE1  : ADC_irq_handler();  break;
  }
}
```

This is the interrupt handler routine.



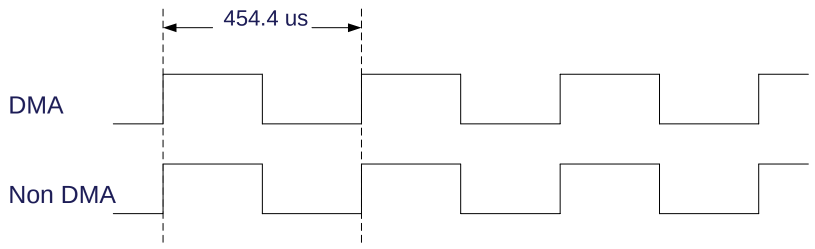
Non DMA ADC Interrupt Handler

```
// ADC Interrupt Handler
// Executed 250,000 per second
void ADC_irq_handler()
{
    {
      ADBUFST &= ~G1_INT_FLAG;  // clear flag
      SPI1DAT0 = ADBUF1;
     }
}
```

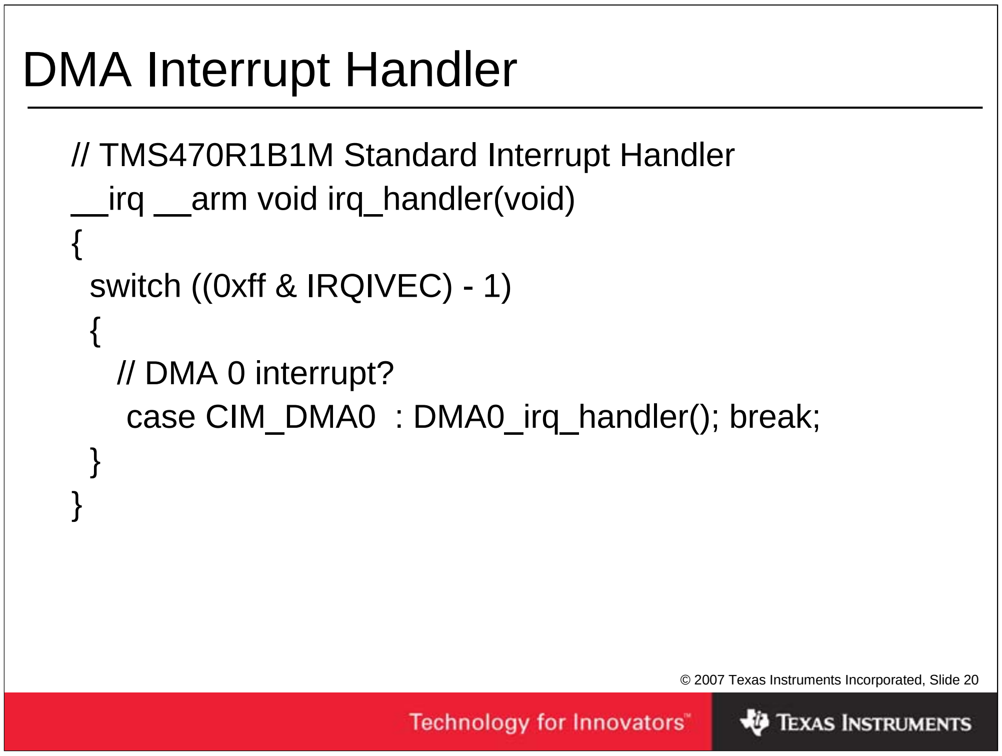
This is the interrupt service routine to handle the data transfer between the ADC and the SPI. It is controlled by the ADC data conversion done signal.

ADC to SPI DMA Transfer

- GIO toggle with out data transfers – 454.4 micro seconds
- GIO toggle with data transfers – 454.4 micro seconds
- 0.0003 % of CPU to facilitate transfer (< 200 CPU clock cycles per second)
- DMA interrupt every 65,535 transfers ~ 4 per second



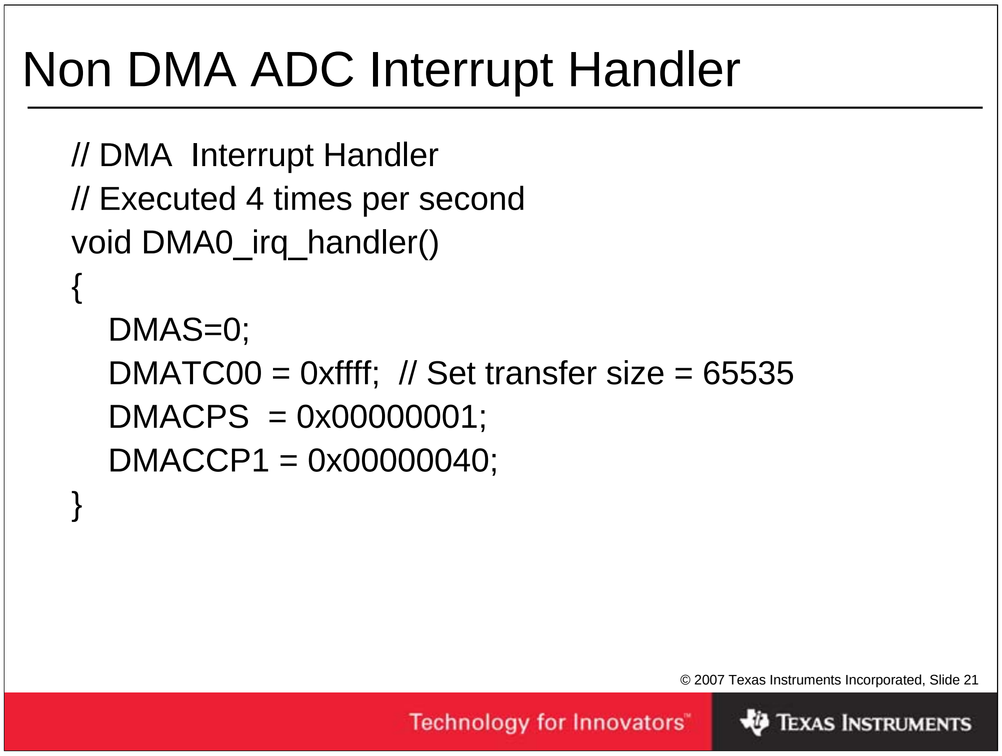
With a CPU clock of 60 MHz the IO toggles with a period of 454.4 micro seconds with no interrupts or DMA for data transfer. When the DMA is enabled the IO toggle appears the same on the oscilloscope. After 65,535 DMA transfers the DMA interrupts the CPU to re initialize the DMA for another 65,535 transfers. It takes about 41 CPU cycles to service the interrupt. This occurs about 4 times per second, thus less than 200 CPU cycles per second for the DMA servicing. This shows very little CPU cycles to service the interrupts verses no data transfers.



DMA Interrupt Handler

```
// TMS470R1B1M Standard Interrupt Handler
__irq __arm void irq_handler(void)
{
  switch ((0xff & IRQIVEC) - 1)
  {
     // DMA 0 interrupt?
      case CIM_DMA0  : DMA0_irq_handler(); break;
  }
}
```

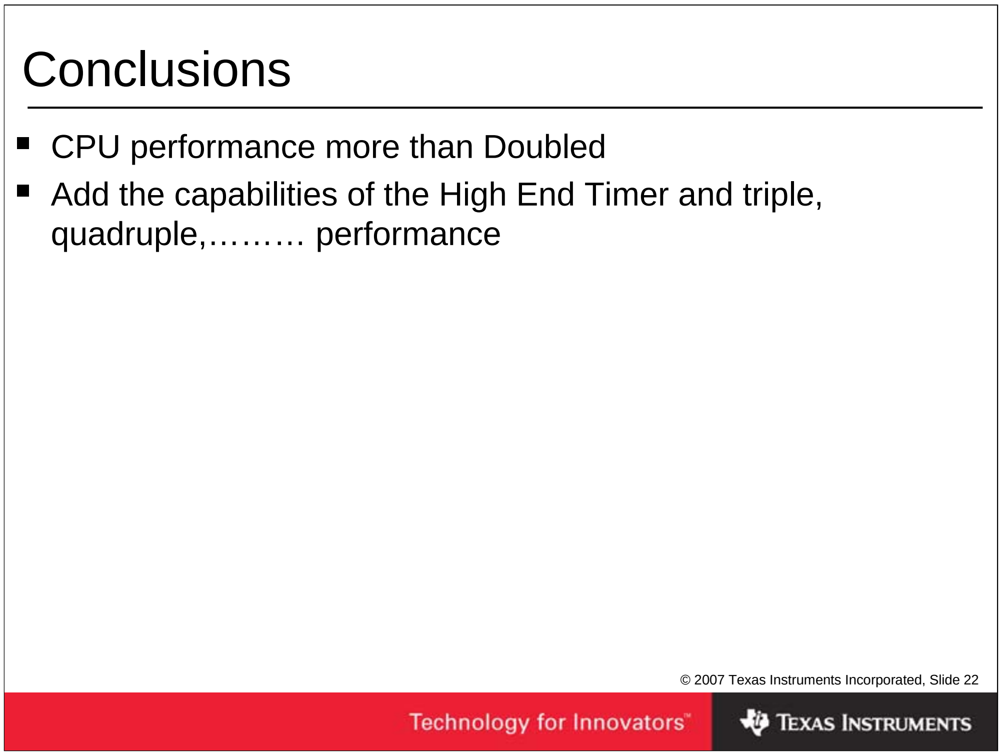
This is the interrupt handler routine.



Non DMA ADC Interrupt Handler

```
// DMA  Interrupt Handler
// Executed 4 times per second
void DMA0_irq_handler()
{
   DMAS=0;
   DMATC00 = 0xffff;  // Set transfer size = 65535
   DMACPS  = 0x00000001;
   DMACCP1 = 0x00000040;
}
```

This is the interrupt service routine to handle the DMA re initialization. It is controlled by the DMA transfer complete signal.



Conclusions

- CPU performance more than Doubled
- Add the capabilities of the High End Timer and triple, quadruple,……… performance

Technology for Innovators™
Texas Instruments

With DMA and other high performance peripherals, the processing power of the CPU can be doubled or more.

Thank You