Embedded systems engineering

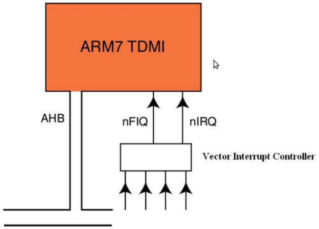
David Kendall

Introduction

- Time-triggered systems and event-triggered systems rely on interrupt handling
- Time-triggered – only one source of interrupt; a timer
- Event-triggered – potentially many sources of interrupt
- Build understanding of interrupt handling by looking in detail at:
    - Installing and executing an interrupt handler (ISR)
    - Configuring a timer as an interrupt source
- Microcontroller – NXP LPC2378

LPC23xx Interrupt Structure

- ARM7 has two external interrupt lines: FIQ and IRQ
- FIQ - typically one interrupt source, so fast
- IRQ - for all other interrupt sources
- VIC - vectored interrupt controller gives hardware support for determining source of IRQ interrupt; speeds up IRQ handling

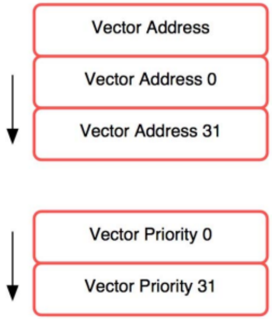


VIC - Vectored Interrupt Controller

- VIC is highly optimised interrupt controller
- Handles on-chip interrupt sources from peripherals
- Each interrupt source connected to VIC on a fixed channel
- Channels can be connected to interrupt lines (FIQ, IRQ) in one of three ways:
    1. as FIQ interrupt
    2. as vectored IRQ interrupt
    3. as non-vectored IRQ interrupt
- Interrupt response time varies for each method
- Method 2 considered in what follows

VIC structure

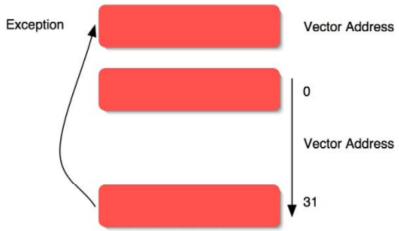
- VIC provides a hardware lookup table for address of each ISR; allows control of priority
- VIC contains 32 slots for vectored addressing
- Each slot contains a Vector Address Register and a Vector Priority Register
- 16 priority levels: 0 - high, 15 - low; initially all 15



Vector Address Register

Address transfer

When an interrupt occurs the contents of the vector address slot associated with the interrupt channel will be transferred by the hardware to the Vector Address Register



Transfer control to the ISR

- VIC transfers address of ISR to Vector Address Register
- ARM7 recognises that IRQ is asserted and begins to execute code from the IRQ vector (`@ 0x00000018`)
- So we need the code at the IRQ vector to fetch the address from the Vector Address Register and start executing the ISR found there

Transfer control to the ISR

Assume

- IRQ vector contains an instruction to transfer control to `IRQ_Handler`
- `VICADDRESS` is the vector address register
- `typedef void (* pVoidFunc_t)(void);`

Then this executes the ISR

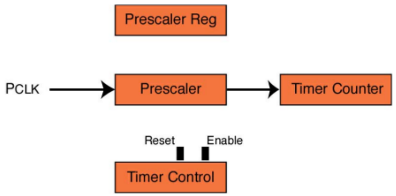
```
__irq __arm void IRQ_Handler (void) {

  if (VICADDRESS != 0) {              // if handler assigned
    (*(pVoidFunc_t)VICADDRESS)();     // call the handler
  }
  VICADDRESS = 0;                     // clear the interrupt
}
```

Install IRQ Handler

```
void vicInstallIRQhandler(pVoidFunc_t pISR, uint32_t priority,
                          uint32_t vicIntSource) {

  // store the handler address in the correct slot in the vector
  *(&VICVECTADDR0 + vicIntSource) = (unsigned long)pISR;

  // set the priority of the interrupt for this slot
  *(&VICVECTPRIORITY0 + vicIntSource) = priority;

  // clear FIQ select bit i.e. assign this interrupt source to IRQ
  VICINTSELECT &= ~(1UL << vicIntSource);

  // enable the interrupt
  VICINTENABLE |= (1UL << vicIntSource);
}
```

LPC23xx Timers

- Four timers all with same structure
- Default clock source is APB peripheral clock (PCLK)
- Prescaler increments on each PCLK tick
- When prescaler value is equal to value in prescaler register, timer counter is incremented by 1 and prescaler is reset

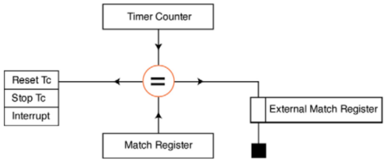


Timer modes

- Timers can be used in
    - capture mode
    - counter mode
    - match mode
- When used in match mode, the timer can be used to trigger some event when the value in the timer counter matches some preset value
- event can be a timer action (reset, stop, interrupt) or external action (e.g. set, clear, reset pin)
- match mode details to follow

Timer match mode

- Each timer has up to four match channels
- Each match channel has a match register containing 32-bit number
- When current value of timer counter matches value in match register an event is triggered



Some Timer0 Registers

Name	Function	Notes
T0TCR	Timer Control	Bit 0: 0 disables counter, 1 enables; Bit 1: 0 counter runs freely, 1 counter is reset
T0PR	Prescale	Value here controls when timer counter is incremented based on PCLK
T0CTCR	Count Control	Bits 0:1, 00 selects timer mode
T0MR0	Match	Write value here to be matched in order to cause event
T0IR	Interrupt	Writing 1 resets register; writing 0 has no effect
T0MCR	Match Control	Bits 0:1, 11 causes interrupt and reset of counter on match event

- see LPC23xx User Manual Chapter 6 for details of timer registers

Managing Timer0

```
void initTimer0(pVoidFunc_t handler, uint32_t ticksPerSec) {
  T0TCR = 0x02;   // reset timer
  T0PR = 0x00;    // set prescaler to 0
  T0CTCR = 0x00; // set mode: every rising PCLK edge
  T0MR0 = getFpclk(TIMER0_PCLK_OFFSET) / ticksPerSec;
  T0IR = 0xff;    // reset all interrupts
  T0MCR = 0x03;   // enable interrupt and reset on match
  vicInstallIRQhandler(handler, 0, VIC_TIMER0);
}

void startTimer0(void) {
  T0TCR = 0x01;   // start timer 0
}

void stopTimer0(void) {
  T0TCR = 0x00;   // stop timer 0
}
```

Acknowledgements

- Trevor Martin, The Insider's Guide to the NXP LPC2300/2400 based Microcontrollers, Hitex (UK) Ltd, 2007