EECS 373
Design of Microprocessor-Based Systems

Robert Dick
University of Michigan

Lecture 7: Interrupts

26 January 2017

Slides inherited from Mark Brehob.

Outline

- Context and review
- Interrupts
  - General characteristics
  - Our Cortex M-3
- Timers
  - General characteristics
  - SmartFusion board

Context and review

- Response to Embedded Systems stand-up routine.
  - Confusion → fear → nervous laughter → relief.
- Hardware vs. software programming.
- APB
  - How to interface with a bus.
  - Need to understand how to do this with a shared bus.
  - Don't need tristate buffers for SmartFusion board.
  - Review handwritten notes and lecture video if still fuzzy.
- Several other topics: volatile, function pointers, weak references.
  - Use the source.

Hardware vs. software programming (again)

- Reasons covering
  - Common sticking point
  - A few students have had trouble with this in lab
- HDL → FPGA
  - Control which functions (gates) are implemented.
  - Control how they are connected.
- Assembly/C → ARM Cortex M-3
  - Control instruction sequences.
  - Control data to load into memory before execution.
- Implications
  - When you write to an MMIO address, the processor/bus controller know how to set and time bus signals. Someone else built that.
  - Your peripheral (SPIO in Lab 3) needs to react to those signals appropriately.

Outline

- ~~Context and review~~
- Interrupts
  - General characteristics
  - Our Cortex M-3
- Timers
  - General characteristics
  - SmartFusion board

Interrupts

Why do these matter?

- Informs a program of some (usually) external event.
- Interrupts execution flow.
- Enables event-driven system design!!!
  - Low-power.
  - Often simpler.

Key questions:

- Where do interrupts come from?
- How do we save state for later continuation?
- How can we ignore interrupts?
- How can we prioritize interrupts?
- How can we share interrupts?

I/O data transfer

Two key questions to determine how data are transferred to/from a non-trivial I/O device.

1. How does the CPU know when data are available?
   a. Polling.
   b. Interrupts.

2. How are data transferred into and out of the device?
   a. Programmed I/O
   b. Direct Memory Access (DMA)

Interrupts

Interrupt (a.k.a. exception or trap) causes CPU to stop executing program and execute an interrupt handler or interrupt service routine (ISR). The ISR does something and then control is returned to the interrupted program.

Interrupts are similar to procedure calls. However,

- can occur between any two instructions and even within some instructions,
- are transparent to the running program (usually),
- are not explicitly requested by the program (typically), and
- call a procedure at an address determined by the type of interrupt, not the program.

Instruction-triggered interrupts

- TLB miss.
- Illegal/unimplemented instruction.
- Divide by 0.
- Trap instruction.
- Names: trap, exception, software interrupt.

Externally triggered interrupts

- External device
- Reset button
- Timer expires
- Power failure
- System error
- Names: interrupt, external interrupt, hardware interrupt

Interrupt process

- Something tells the processor there is an interrupt, e.g., via an input pin.
- Processor transfers control to code that needs to be executed through interrupt vector or jump table.
- ISR executes.
- Resumes prior program at same location.
- Doing this right is complex.

Interrupts complicate processor design

- Which ISR to call?
- How to resume program when done?
  - Instruction pointer? Other state?
- What about partially executed instructions in the pipeline?
- What if we get an interrupt while we are processing our interrupt?
  - What if we are in a “critical section?”

Where

- If you know the interrupt source.
  - Interrupt vector.
  - Jump table.
- If not.
  - Must poll all sources to find out.

Returning

- Need to store the return address somewhere.
  - Stack would involve a load/store that might cause another interrupt.
  - Dedicated register.
    - What if there is another interrupt?

Implications of architectural optimizations

- Out-of-order execution
  - If any state of a "too fast" instruction made its way out of the processor before an interrupt, system state corrupted.
- Need to clean things up before/in ISR.

Nested interrupts

- Just handle it.
  - If a dedicated interrupt return IP register is being used, how many do we need?
  - What if the ISR is half-way through a precisely times bus transaction?
- Ignore it: Bad if it is important.
- Prioritize.
  - Take more important interrupts.
  - Ignore the rest
  - Still have dedicated register problems.
  - Have to consider possibility of ISR failing due to timing problems.

Critical section

- Ignore less important interrupts.
- Take more important interrupts.
- Avoid causing exceptions in interrupt code.
- Keep as short as possible.
  - E.g., write a value to memory that informs the program of something.
  - Program deals with it at a good time.

Example: generally bad

```
void isr(void) {
   Do something complex/slow.
}
```

Example: generally good

```
void isr(void) {
   ++(*button_pressed);
}

int superloop(void) {
   while (1) {
      if (*button_pressed) {
         --(*button_pressed);
         button_service();
      }
      Do other stuff, like AI.
      Could also sleep.
   }
}
```


Outline

- ~~Context and review~~
- Interrupts
  - ~~General characteristics~~
  - Our Cortex M-3
- Timers
  - General characteristics
  - SmartFusion board

Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard fault	−1	All fault conditions if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations
5	Bus fault	Programmable	Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage fault	Programmable	Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	—
11	SVC	Programmable	Supervisor Call
12	Debug monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	—
14	PendSV	Programmable	Pendable Service Call
15	SYSTICK	Programmable	System Tick Timer

Table 7.2 List of External Interrupts

Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
...	...	...
255	External Interrupt #239	Programmable

SmartFusion interrupt sources

Table 1-5 • SmartFusion Interrupt Sources

Cortex-M3 NVIC Input	IRQ Label	IRQ Source
NMI	WDOGTIMEOUT_IRQ	WATCHDOG
INTISR[0]	WDOGWAKEUP_IRQ	WATCHDOG
INTISR[1]	BROWNOUT1_5V_IRQ	VR/PSM
INTISR[2]	BROWNOUT3_3V_IRQ	VR/PSM
INTISR[3]	RTCMATCHEVENT_IRQ	RTC
INTISR[4]	PU_N_IRQ	RTC
INTISR[5]	EMAC_IRQ	Ethernet MAC
INTISR[6]	M3_IAP_IRQ	IAP
INTISR[7]	ENVM_0_IRQ	ENVM Controller
INTISR[8]	ENVM_1_IRQ	ENVM Controller
INTISR[9]	DMA_IRQ	Peripheral DMA
INTISR[10]	UART_0_IRQ	UART_0
INTISR[11]	UART_1_IRQ	UART_1
INTISR[12]	SPI_0_IRQ	SPI_0
INTISR[13]	SPI_1_IRQ	SPI_1
INTISR[14]	I2C_0_IRQ	I2C_0
INTISR[15]	I2C_0_SMBALERT_IRQ	I2C_0
INTISR[16]	I2C_0_SMBSUS_IRQ	I2C_0
INTISR[17]	I2C_1_IRQ	I2C_1
INTISR[18]	I2C_1_SMBALERT_IRQ	I2C_1
INTISR[19]	I2C_1_SMBSUS_IRQ	I2C_1
INTISR[20]	TIMER_1_IRQ	TIMER
INTISR[21]	TIMER_2_IRQ	TIMER
INTISR[22]	PLLLOCK_IRQ	MSS_CCC
INTISR[23]	PLLLOCKLOST_IRQ	MSS_CCC
INTISR[24]	ABM_ERROR_IRQ	AHB BUS MATRIX
INTISR[25]	Reserved	Reserved
INTISR[26]	Reserved	Reserved
INTISR[27]	Reserved	Reserved
INTISR[28]	Reserved	Reserved
INTISR[29]	Reserved	Reserved
INTISR[30]	Reserved	Reserved
INTISR[31]	FAB_IRQ	FABRIC INTERFACE
INTISR[32]	GPIO_0_IRQ	GPIO
INTISR[33]	GPIO_1_IRQ	GPIO
INTISR[34]	GPIO_2_IRQ	GPIO

GPIO_3_IRQ to GPIO_31_IRQ cut

INTISR[64]	ACE_PC0_FLAG0_IRQ	ACE
INTISR[65]	ACE_PC0_FLAG1_IRQ	ACE
INTISR[66]	ACE_PC0_FLAG2_IRQ	ACE
INTISR[67]	ACE_PC0_FLAG3_IRQ	ACE
INTISR[68]	ACE_PC1_FLAG0_IRQ	ACE
INTISR[69]	ACE_PC1_FLAG1_IRQ	ACE
INTISR[70]	ACE_PC1_FLAG2_IRQ	ACE
INTISR[71]	ACE_PC1_FLAG3_IRQ	ACE
INTISR[72]	ACE_PC2_FLAG0_IRQ	ACE
INTISR[73]	ACE_PC2_FLAG1_IRQ	ACE
INTISR[74]	ACE_PC2_FLAG2_IRQ	ACE
INTISR[75]	ACE_PC2_FLAG3_IRQ	ACE
INTISR[76]	ACE_ADC0_DATAVALID_IRQ	ACE
INTISR[77]	ACE_ADC1_DATAVALID_IRQ	ACE
INTISR[78]	ACE_ADC2_DATAVALID_IRQ	ACE
INTISR[79]	ACE_ADC0_CALDONE_IRQ	ACE
INTISR[80]	ACE_ADC1_CALDONE_IRQ	ACE
INTISR[81]	ACE_ADC2_CALDONE_IRQ	ACE
INTISR[82]	ACE_ADC0_CALSTART_IRQ	ACE
INTISR[83]	ACE_ADC1_CALSTART_IRQ	ACE
INTISR[84]	ACE_ADC2_CALSTART_IRQ	ACE
INTISR[85]	ACE_COMP0_FALL_IRQ	ACE
INTISR[86]	ACE_COMP1_FALL_IRQ	ACE
INTISR[87]	ACE_COMP2_FALL_IRQ	ACE
INTISR[88]	ACE_COMP3_FALL_IRQ	ACE
INTISR[89]	ACE_COMP4_FALL_IRQ	ACE
INTISR[90]	ACE_COMP5_FALL_IRQ	ACE
INTISR[91]	ACE_COMP6_FALL_IRQ	ACE
INTISR[92]	ACE_COMP7_FALL_IRQ	ACE
INTISR[93]	ACE_COMP8_FALL_IRQ	ACE
INTISR[94]	ACE_COMP9_FALL_IRQ	ACE
INTISR[95]	ACE_COMP10_FALL_IRQ	ACE

54 more ACE specific interrupts

Interrupt vectors (in startup_a2fxxxm3.s found in CMSIS, startup_gcc)

```
g_pfnVectors:
    .word  _estack
    .word  Reset_Handler
    .word  NMI_Handler
    .word  HardFault_Handler
    .word  MemManage_Handler
    .word  BusFault_Handler
    .word  UsageFault_Handler
    .word  0
    .word  0
    .word  0
    .word  0
    .word  SVC_Handler
    .word  DebugMon_Handler
    .word  0
    .word  PendSV_Handler
    .word  SysTick_Handler
    .word  WdogWakeup_IRQHandler
    .word  BrownOut_1_5V_IRQHandler
    .word  BrownOut_3_3V_IRQHandler
.............. (they continue)
```

Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard fault	−1	All fault conditions if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations
5	Bus fault	Programmable	Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage fault	Programmable	Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	—
11	SVC	Programmable	Supervisor Call
12	Debug monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	—
14	PendSV	Programmable	Pendable Service Call
15	SYSTICK	Programmable	System Tick Timer

Table 7.2 List of External Interrupts

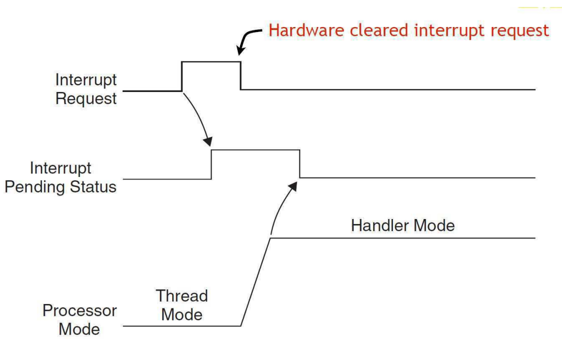
Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
...	...	...
255	External Interrupt #239	Programmable

Interrupt handlers

```
g_pfnVectors:
    .word  _estack
    .word  Reset_Handler
    .word  NMI_Handler
    .word  HardFault_Handler
    .word  MemManage_Handler
    .word  BusFault_Handler
    .word  UsageFault_Handler
    .word  0
    .word  0
```

```
/*==============================================
 * Reset_Handler
 */
     .global Reset_Handler
     .type   Reset_Handler, %function
Reset_Handler:
_start:
```

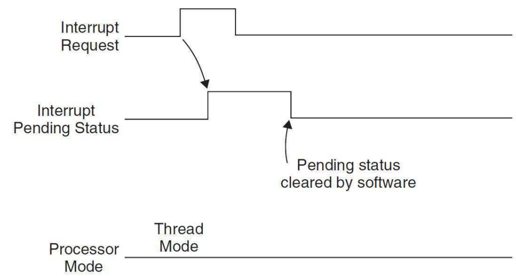
Pending interrupts



The normal case. Once Interrupt request is seen, processor puts it in "pending" state even if hardware drops the request.
IPS is cleared by the hardware once we jump to the ISR.

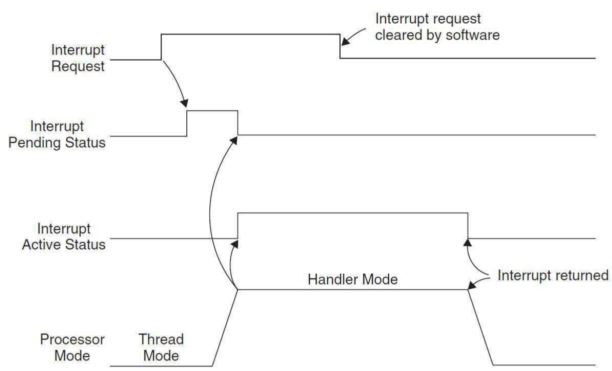
This figure and those following are from *The Definitive Guide to the ARM Cortex-M3, Section 7.4*

Untaken interrupts

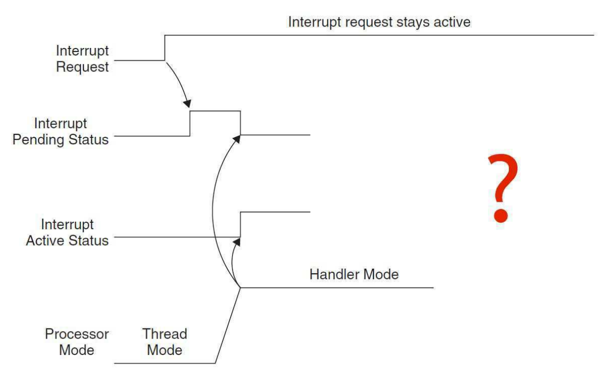


In this case, the processor never took the interrupt because we cleared the IPS by hand (via a memory-mapped I/O register)

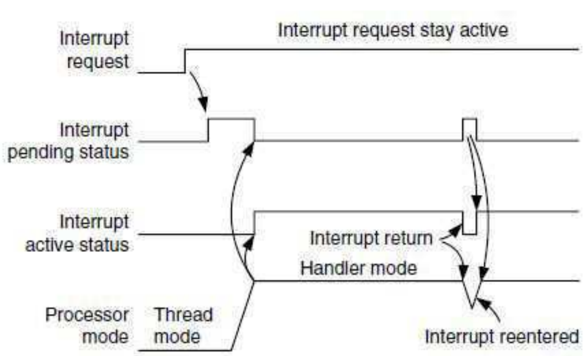
Active Status set during handler execution



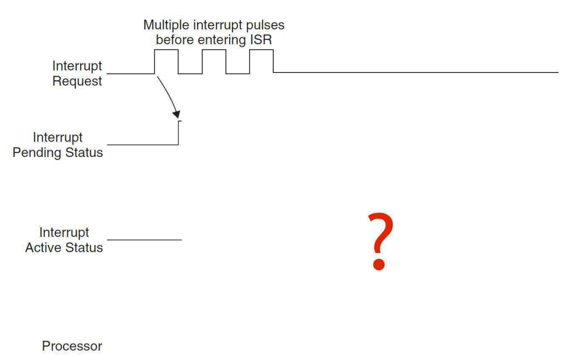
Interrupt Request not Cleared



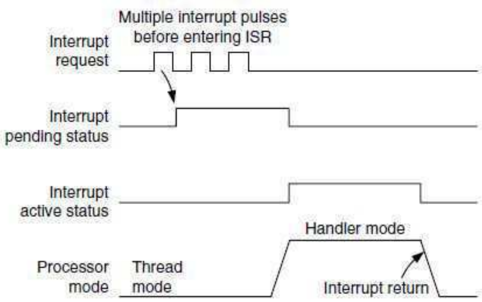
Answer



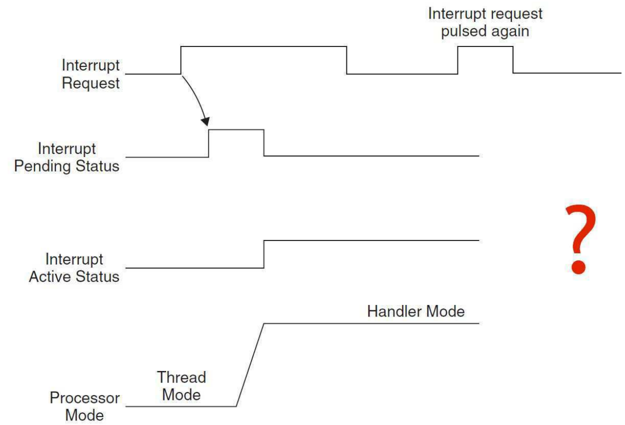
Interrupt pulses before entering ISR



Answer



New Interrupt Request after Pending Cleared



Tail chaining

- Processor can serve multiple interrupts without returning to program.
- Improves response latency.
  - No need for state save/restore.

Configuring the NVIC

- Interrupt Set Enable and Clear Enable
  - 0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C

0xE000E100	SETENA0	R/W	0	Enable for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E180	CLRENA0	R/W	0	Clear enable for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 … bit[31] for interrupt #31 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status

Configuring the NVIC (2)

- Set Pending & Clear Pending
  - 0xE000E200-0xE000E21C, 0xE000E280-0xE000E29C

0xE000E200	SETPEND0	R/W	0	Pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E280	CLRPEND0	R/W	0	Clear pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status

Configuring the NVIC (3)

- Interrupt Active Status Register
  - 0xE000E300-0xE000E31C

Address	Name	Type	Reset Value	Description
0xE000E300	ACTIVE0	R	0	Active status for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 … bit[31] for interrupt #31
0xE000E304	ACTIVE1	R	0	Active status for external interrupt #32–63
…	–	–	–	–

Interrupt priorities

- If multiple interrupts arrive at same time, prioritize.
- 3 fixed highest priorities.
- Up to 256 programmable priorities and 128 preemption levels.
- Particular processors support a subset of priorities.
- SmartFusion supports 32 priorities: five highest bits.
- 0, 8, 16, 32, 24, 32, …
- Higher priorities preempt lower.
- Priority can be sub-divided into groups.
  - Splits register into preempt priority and subpriority.
  - Subpriority used if two interrupts with same preempt priority arrive at same time.

Interrupt Priority (2)

- Interrupt Priority Level Registers
  - 0xE000E400-0xE000E4EF

Address	Name	Type	Reset Value	Description
0xE000E400	PRI_0	R/W	0 (8-bit)	Priority-level external interrupt #0
0xE000E401	PRI_1	R/W	0 (8-bit)	Priority-level external interrupt #1
…	–	–	–	–
0xE000E41F	PRI_31	R/W	0 (8-bit)	Priority-level external interrupt #31
…	–	–	–	–

Preemption Priority and Subpriority

Priority Group	Preempt Priority Field	Subpriority Field
0	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]
5	Bit [7:6]	Bit [5:0]
6	Bit [7]	Bit [6:0]
7	None	Bit [7:0]

Use PRIGROUP field to control split.

Application Interrupt and Reset Control Register (Address 0xE000ED0C)

Bits	Name	Type	Reset Value	Description
31:16	VECTKEY	R/W	–	Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05
15	ENDIANNESS	R	–	Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset
10:8	PRIGROUP	R/W	0	Priority group
2	SYSRESETREQ	W	–	Requests chip control logic to generate a reset
1	VECTCLRACTIVE	W	–	Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer)
0	VECTRESET	W	–	Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor

PRIMASK, FAULTMASK, and BASEPRI

- What if we quickly want to disable all interrupts?

- Write 1 into PRIMASK to disable all interrupt except NMI
  - MOV R0, #1
  - MSR PRIMASK, R0
- Write 0 into PRIMASK to enable all interrupts
- FAULTMASK is the same as PRIMASK, but also blocks hard fault (priority -1)

- What if we want to disable all interrupts below a certain priority?
- Write priority into BASEPRI
  - MOV R0, #0x60
  - MSR BASEPRI, R0

Masking

B1.4.3 The special-purpose mask registers

There are three special-purpose registers which are used for the purpose of priority boosting. Their function is explained in detail in *Execution priority and priority boosting within the core* on page B1-18:

- the exception mask register (PRIMASK) which has a 1-bit value
- the base priority mask (BASEPRI) which has an 8-bit value
- the fault mask (FAULTMASK) which has a 1-bit value.

All mask registers are cleared on reset. All unprivileged writes are ignored.

The formats of the mask registers are illustrated in Table B1-4.

Table B1-4 The special-purpose mask registers

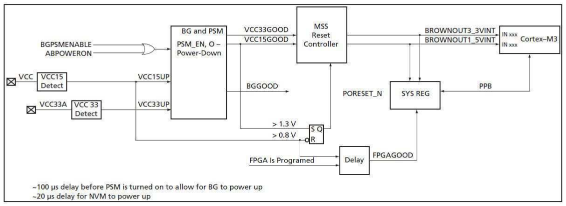
	31 … 8	7 … 1	0
PRIMASK	RESERVED		PM
FAULTMASK	RESERVED		FM
BASEPRI	RESERVED	BASEPRI	

Interrupt Service Routines

1. Automatic saving of registers upon exception
   - PC, PSR, R0-R3, R12, LR pushed on the stack
2. While bus busy, fetch exception vector
3. Update SP to new location
4. Update IPSR (low part of PSR) with new exception number
5. Set PC to vector handler
6. Update LR to special value EXC_RETURN

- Several other NVIC registers get updated
- Latency: as short as 12 cycles

Example of complexity: the Reset Interrupt



1) No power.
2) System is held in RESET as long as VCC15 < 0.8V.
   a) In reset: registers forced to default.
   b) RC-Osc begins to oscillate.
   c) MSS_CCC drives RC-Osc/4 into FCLK.
   d) PORESET_N is held low.
3) Once VCC15GOOD, PORESET_N goes high.
   a) MSS reads from eNVM address 0x0 and 0x4.

The xPSR register layout

The APSR, IPSR and EPSR registers are allocated as mutually exclusive bitfields within a 32-bit register. The combination of the APSR, IPSR and EPSR registers is referred to as the xPSR register.

Table B1-2 The xPSR register layout

	31	30	29	28	27	26 25	24	23 … 16	15 … 10	9	8 … 0
APSR	N	Z	C	V	Q						
IPSR											0 or Exception Number
EPSR						ICI/IT	T		ICI/IT	a	

WFI: Wait For Interrupt

- Puts processor in low-power mode and waits for interrupt.
- Why?

Two stacks? MSP and PSP

- OS always uses MSP.
- Can configure processor so program uses PSP.
- Makes it harder for application code to corrupt OS/superloop state.

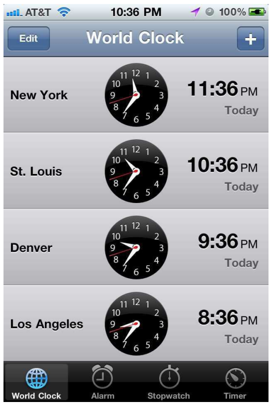
Outline

- ~~Context and review~~
- ~~Interrupts~~
  - ~~General characteristics~~
  - ~~Our Cortex M-3~~
- Timers
  - General characteristics
  - SmartFusion board

Timers

- Why they matter?
- Avoid pitfalls of loop-based delays.
  - Waste power.
  - Prevent other useful work from being done.
- Why they are complex?
  - Span HW/SW boundary.

iPhone Clock App



- World Clock – display real time in multiple time zones
- Alarm – alarm at certain (later) time(s).
- Stopwatch – measure elapsed time of an event.
- Timer – count down time and notify when count becomes zero.

Motor and light Control

- Servo motors – PWM signal provides control signal.
- DC motors – PWM signals control power delivery.
- RGB LEDs – PWM signals allow dimming through current-mode control.

Methods from Android SystemClock

Public Methods	
static long	currentThreadTimeMillis() Returns milliseconds running in the current thread.
static long	elapsedRealtime() Returns milliseconds since boot, including time spent in sleep.
static long	elapsedRealtimeNanos() Returns nanoseconds since boot, including time spent in sleep.
static boolean	setCurrentTimeMillis(long millis) Sets the current wall time, in milliseconds.
static void	sleep(long ms) Waits a given number of milliseconds (of uptimeMillis) before returning.
static long	uptimeMillis() Returns milliseconds since boot, not counting time spent in deep sleep.

Standard C library's <time.h> header file

Library Functions

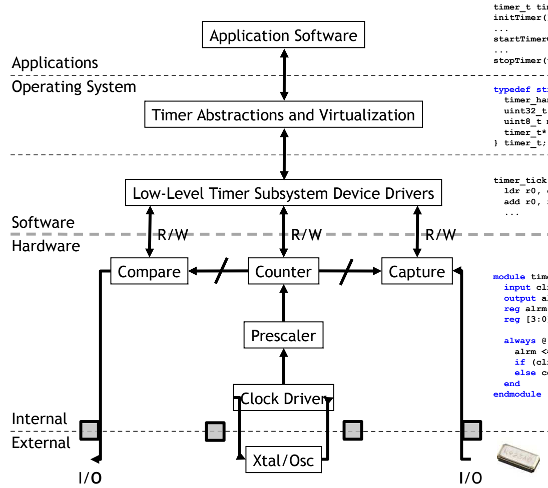
Following are the functions defined in the header time.h:

S.N.	Function & Description
1	char *asctime(const struct tm *timeptr) Returns a pointer to a string which represents the day and time of the structure timeptr.
2	clock_t clock(void) Returns the processor clock time used since the beginning of an implementation-defined era (normally the beginning of the program).
3	char *ctime(const time_t *timer) Returns a string representing the localtime based on the argument timer.
4	double difftime(time_t time1, time_t time2) Returns the difference of seconds between time1 and time2 (time1-time2).
5	struct tm *gmtime(const time_t *timer) The value of timer is broken up into the structure tm and expressed in Coordinated Universal Time (UTC) also known as Greenwich Mean Time (GMT).
6	struct tm *localtime(const time_t *timer) The value of timer is broken up into the structure tm and expressed in the local time zone.
7	time_t mktime(struct tm *timeptr) Converts the structure pointed to by timeptr into a time_t value according to the local time zone.
8	size_t strftime(char *str, size_t maxsize, const char *format, const struct tm *timeptr) Formats the time represented in the structure timeptr according to the formatting rules defined in format and stored into str.
9	time_t time(time_t *timer) Calculates the current calender time and encodes it into time_t format.

Standard C library's <time.h> header file: struct tm

```
struct tm {
   int tm_sec;         /* seconds,  range 0 to 59          */
   int tm_min;         /* minutes, range 0 to 59           */
   int tm_hour;        /* hours, range 0 to 23             */
   int tm_mday;        /* day of the month, range 1 to 31  */
   int tm_mon;         /* month, range 0 to 11             */
   int tm_year;        /* The number of years since 1900   */
   int tm_wday;        /* day of the week, range 0 to 6    */
   int tm_yday;        /* day in the year, range 0 to 365  */
   int tm_isdst;       /* daylight saving time             */
};
```


Anatomy of a timer system



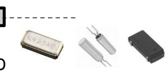

```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

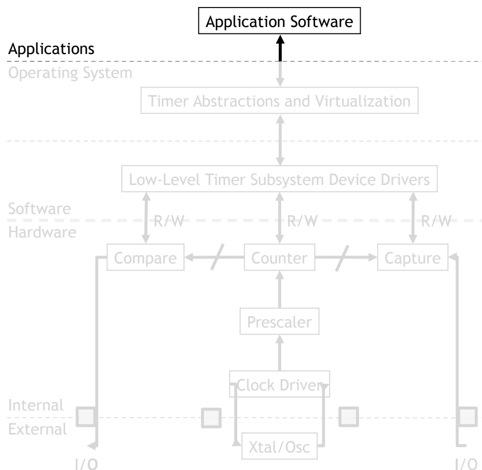
```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```



Anatomy of a timer system




```
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```


Timer requirements

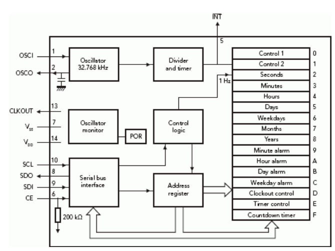
- Wall clock date & time
  - Date: Month, Day, Year
  - Time: HH:MM:SS:mmm
  - Provided by a “real-time clock” or RTC
- Alarm: do something (call code) at certain time later
  - Later could be a delay from now (e.g., Δt)
  - Later could be actual time (e.g., today at 3pm)
- Stopwatch: measure (elapsed) time of an event
  - Instead of pushbuttons, could be function calls or
  - Hardware signals outside the processor

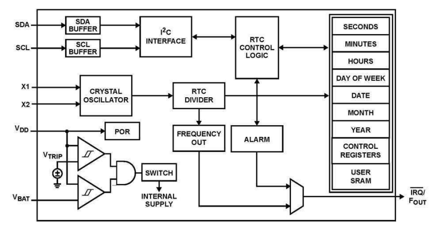
Timer requirements

- Wall clock
  - datetime_t getDateTime()
- Alarm
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: measure (elapsed) time of an event
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

    LDR R1, [R0, #0] % R0=timer address

Wall Clock from a Real-Time Clock (RTC)





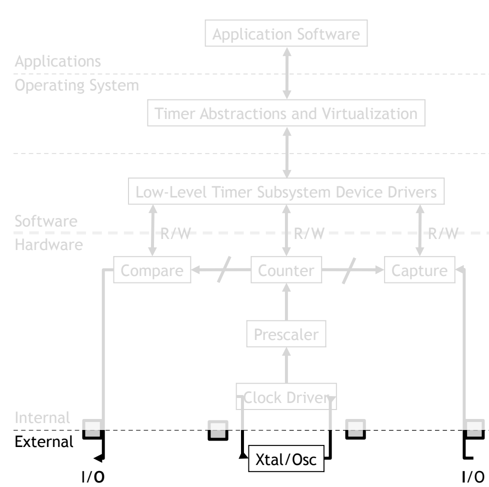
- Often a separate module
- Built with registers for
  - Years, Months, Days
  - Hours, Mins, Seconds
- Alarms: hour, min, day
- Accessed via
  - Memory-mapped I/O
  - Serial bus (I2C, SPI)

Timer requirements

- Wall clock
  - datetime_t getDateTime()
- Alarm
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: measure (elapsed) time of an event
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

    LDR R1, [R0, #0] % R0=timer address

Anatomy of a timer system




```
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

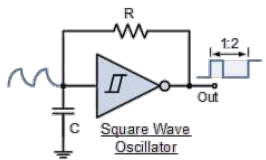
```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```

Oscillators – RC



Oscillators – Crystal

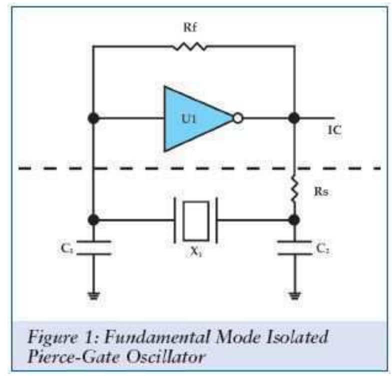
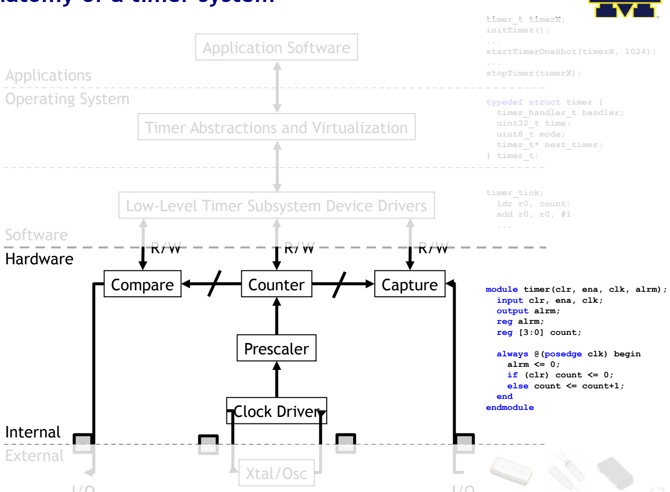


Figure 1: Fundamental Mode Isolated Pierce-Gate Oscillator

Anatomy of a timer system



Timer requirements

- Wall clock
  - datetime_t getDateTime()
- Alarm
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: measure (elapsed) time of an event
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

    LDR R1, [R0, #0]   % R0=timer address

Timer applications

There are two basic activities one wants timers for:

- Measure how long something takes
  - "Capture"
- Have something happen once or every X time period
  - "Compare"

Example # 1: Capture

- Fan
  - Say you have a fan spinning and you want to know how fast it is spinning. One way to do that is to have it throw an interrupt every time it completes a rotation.
    - Right idea, but might take a while to process the interrupt, heavily loaded system might see slower fan than actually exists.
    - This could be bad.
  - Solution? Have the timer note *immediately* how long it took and then generate the interrupt. Also restart timer immediately.
- Same issue would exist in a car when measuring speed of a wheel turning (for speedometer or anti-lock brakes).

Example # 2: Compare

- Driving a DC motor via PWM.
  - Motors turn at a speed determined by the voltage applied.
    - Doing this in analog can be hard.
      - Need to get analog out of our processor
      - Need to amplify signal in a linear way (op-amp?)
        - » Generally prefer just switching between “Max” and“Off” quickly.
      - Average is good enough.
      - Now don’t need linear amplifier—just “on” and “off”. (transistor)
  - Need a signal with a certain duty cycle and frequency.
    - That is % of time high.

Servo motor control: class exercise

- Assume 1 MHz CLK
- Design “high-level” circuit to
  - Generate 1.52 ms pulse
  - Every 6 ms
  - Repeat
- How would we generalize this?

Outline

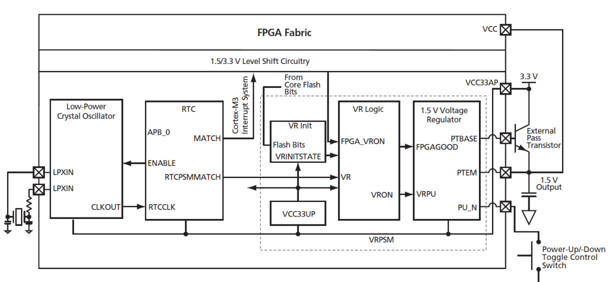
- ~~Context and review~~
- ~~Interrupts~~
  - ~~General characteristics~~
  - ~~Our Cortex M-3~~
- Timers
  - ~~General characteristics~~
  - SmartFusion board

Timers on the SmartFusion

- SysTick Timer
  - ARM requires every Cortex-M3 to have this timer.
  - 24-bit count-down timer to generate system ticks.
  - Has own interrupt.
  - Clocked by FCLK with optional programmable divider.
- See Actel SmartFusion MSS User Guide for register definitions.

Timers on the SmartFusion

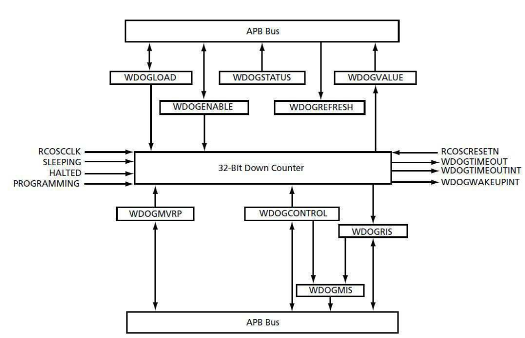
- Real-Time Counter (RTC) System
  - Clocked from 32 kHz low-power crystal
  - Automatic switching to battery power if necessary
  - Can put rest of the SmartFusion to standby or sleep to reduce power
  - 40-bit match register clocked by 32.768 kHz divided by 128 (256 Hz)



http://www.actel.com/documents/SmartFusion_MSS_UG.pdf

Timers on the SmartFusion

- Watchdog Timer
  - 32-bit down counter
  - Either reset system or NMI Interrupt if it reaches 0!

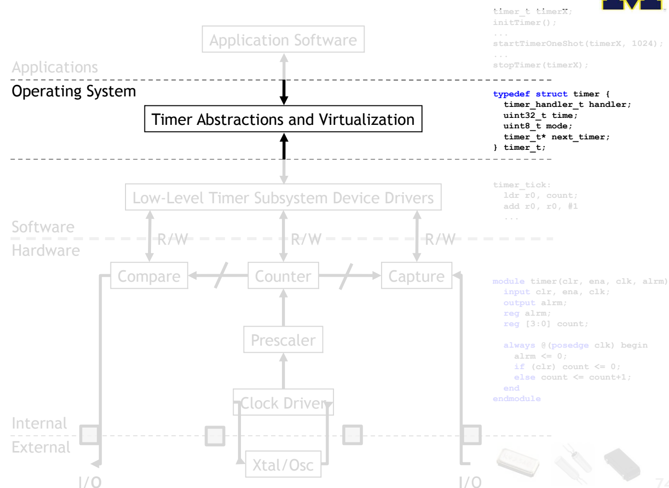


Timers on the SmartFusion

- System timer
  - "The System Timer consists of two programmable 32-bit decrementing counters that generate interrupts to the ARM® Cortex™-M3 and FPGA fabric. Each counter has two possible modes of operation: Periodic mode or One-Shot mode. The two timers can be concatenated to create a 64-bit timer with Periodic and One-Shot modes. The two 32-bit timers are identical"

http://www.actel.com/documents/SmartFusion_MSS_UG.pdf

Anatomy of a timer system



Virtual timers

- Can we use more timers than exist in hardware?
- Yes. Use hardware timers as a foundation for software-controlled virtual timers.
- Maybe we have 10 events we might want to generate.
- Make a list of them and set the timer to go off for the first one.
- Repeat.

Problems?

- Only works for "compare" timer uses.
- Will result in slower ISR response time.
  - May not care, could just schedule sooner.