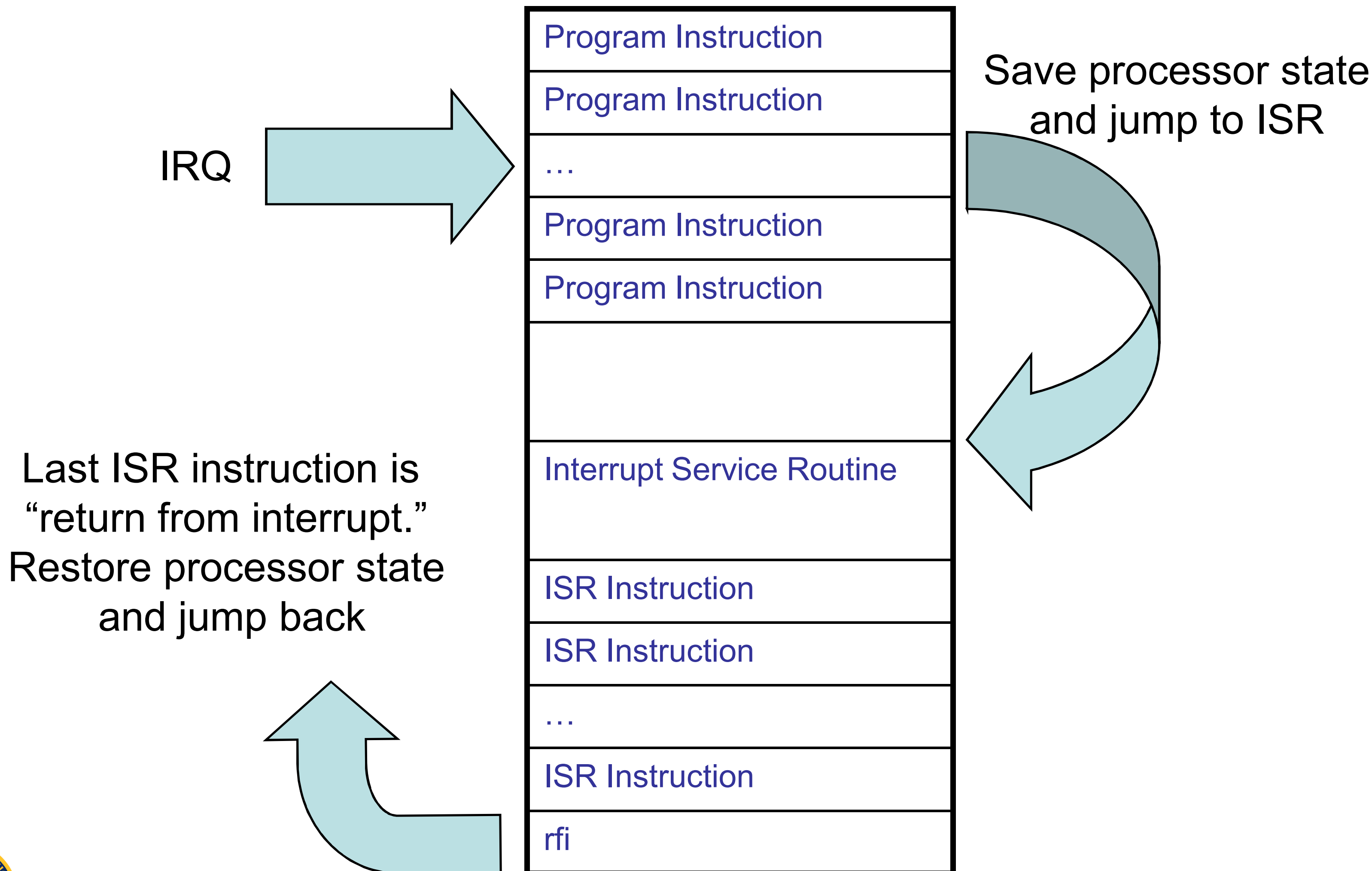
Lab 5: Interrupts, Timing, and Frequency Analysis of PWM Signals

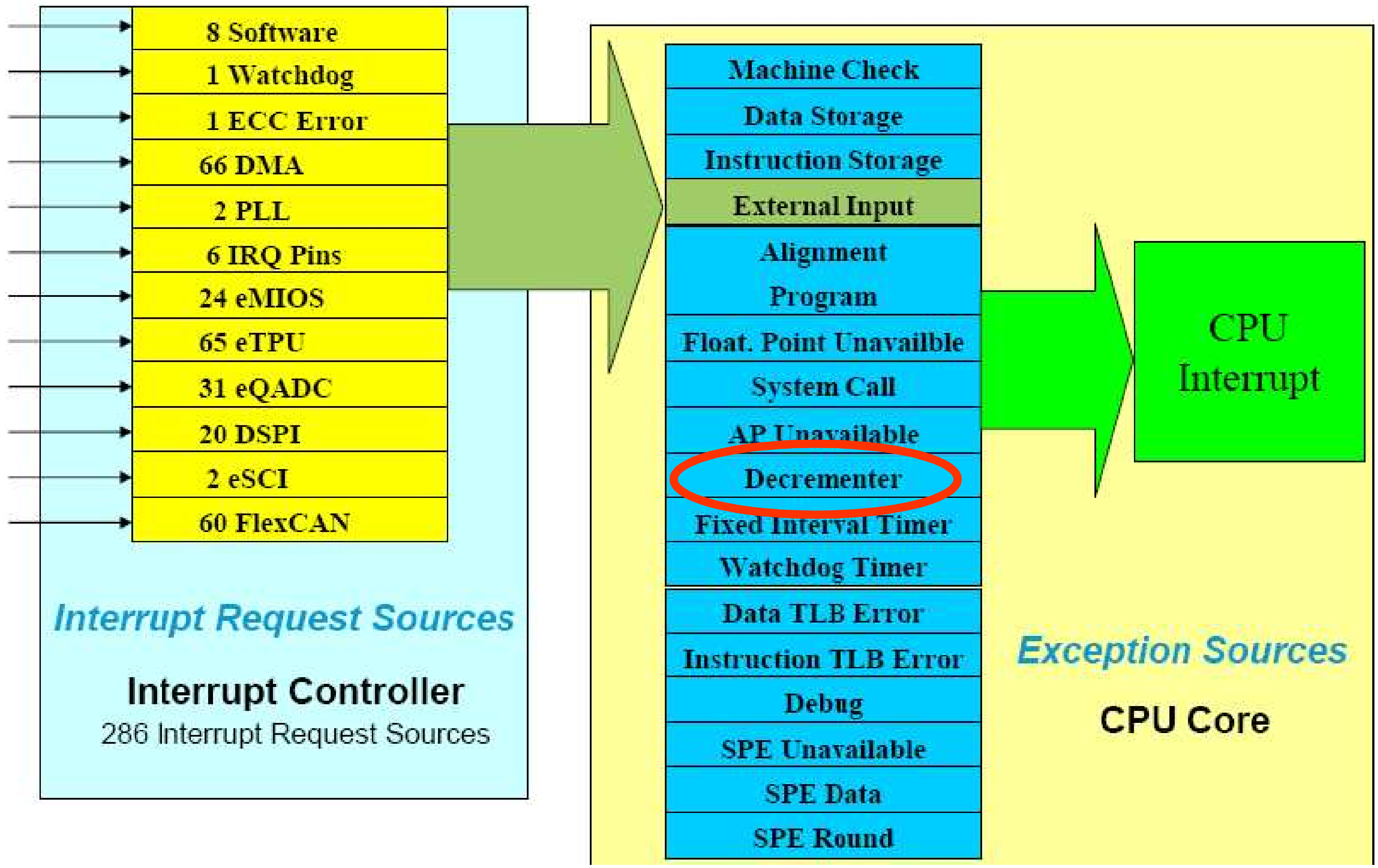
Lab 5: Interrupts and Timing

- Thus far, we have not worried about *time* in our real-time code
  - Almost all real-time code involves sampling (recall our discussion about sampling and aliasing)
- MPC5553 incorporates several timers that can be configured to generate periodic *interrupt requests* (IRQ)
  - Application code stops what its doing and control transfers to an *interrupt service routine* (ISR); ISR may sample a signal (using the eQADC, for example) or generate a signal (using the eMIOS or eTPU, for example)
  - Interrupt requests may also be generated by an external event

Interrupt Service

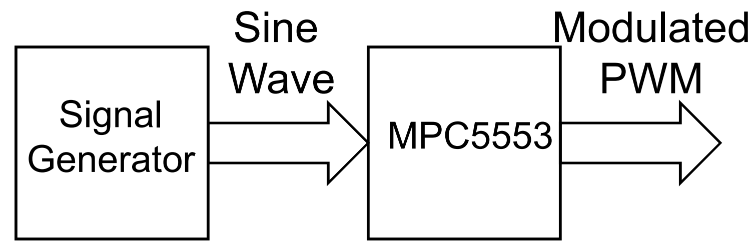


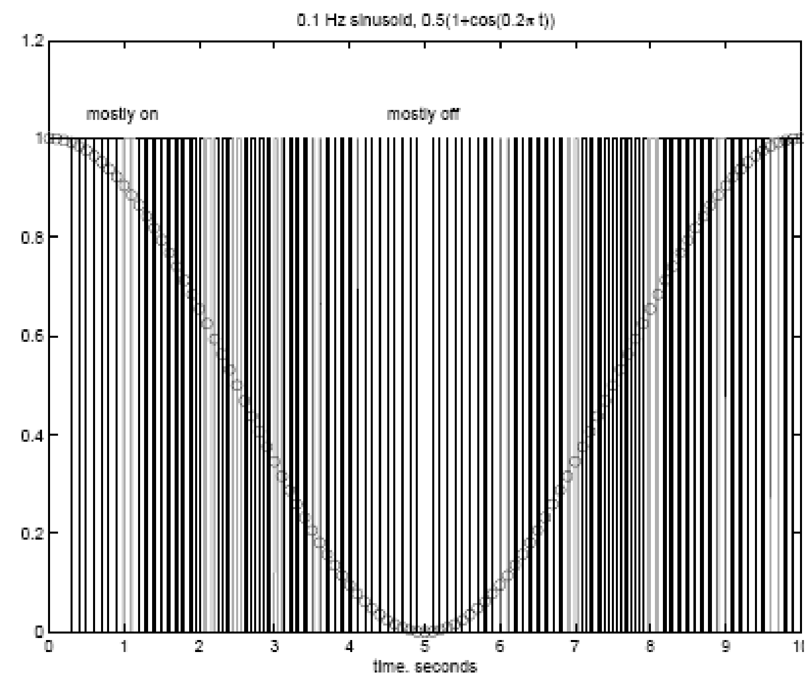
MPC5553 IRQ and Exception Sources



Lab 5: Basic Idea (see Lecture 6)

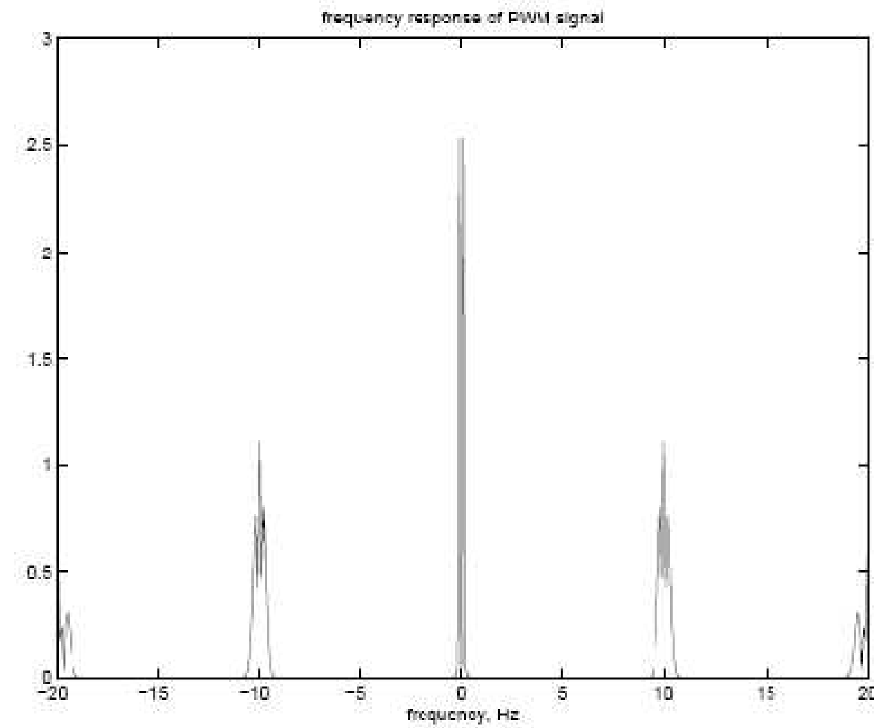
- Generate a sine wave which will be periodically sampled
  - Signal generator
  - "C" sine function
  - Look-up table
- Create a PWM signal with duty cycle equivalent to the sampled value (0-100% in our example)
- Appropriately filter the modulated PWM signal to recover the sine wave
- What's the point?
  - Learn to use the DEC timer
  - RT S/W overhead issues
  - Demonstrate motor response to PWM





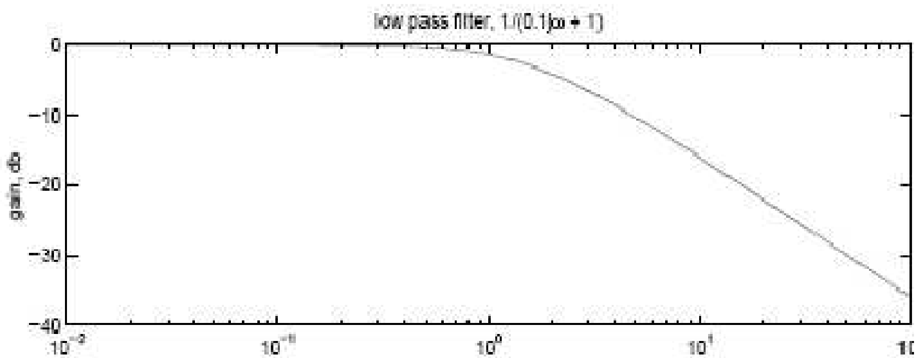
Lab 5: Basic Idea (see Lecture 6)

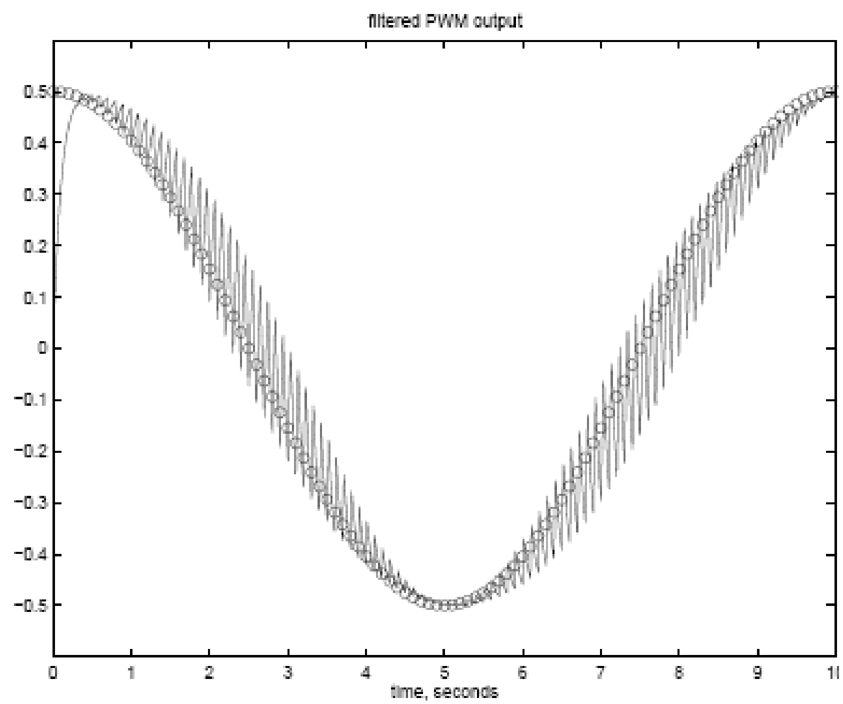
- Suppose we sample at 0.1 second intervals and generate a 10 Hz PWM (we'll use much higher frequency and faster sampling in the lab)
- Neglecting the DC bias (rescale from 0 to10 volts to –5 to 5 volts), frequency spectrum of PWM signal has components at +/- 0.1 Hz, and multiples of the 10 Hz switching frequency



Lab 5: Basic Idea (see Lecture 6)

- Now all we have to do is low-pass filter the high frequency components of our signal to reconstruct the original sine wave
- Filter with unity gain at 0.1 Hz; very small gain at 10 Hz





Lab 5: DEC (Decrementer)

- Timers are not peripheral devices like the eMIOS or eTPU
  - Part of the "core" processor
  - See "e200z6 PowerPC™ Core' Reference Manual for details
- Fixed Interval Timer
- "Watchdog" Timer
- Decrement Timer
  - General software timer
  - 32-bit register counts down and generates an IRQ
  - Automatically reloaded from DECAR register

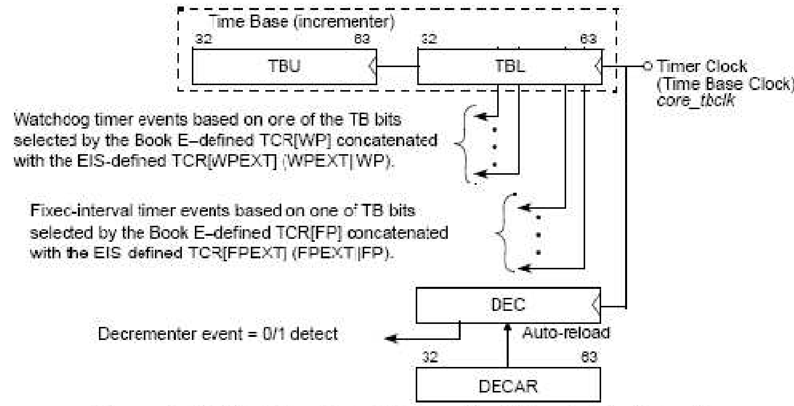


Figure 2-23. Relationship of Timer Facilities to the Time Base

Lab 5: Software

- Use I/O software you developed in labs 3 and 4
  - `qadc.h` and `qadc.c`
    - Read the input sine wave
  - `mios.h` and `mios.c`
    - Generate the PWM signal
- Code required to initialize the DEC and set up interrupt service routines
  - `isr.h` and `isr.c`
  - Routines are written for you
- Three ISRs required
  - Read duty cycle from signal generator
  - Calculate duty cycle using C function
  - Calculate duty cycle using look-up table

isr.c Initializes Decrementer

```
/* from example by S.Mihalik see e200z6 Reference Manual for register defs */
asm void init_DEC(long count) {
#pragma unused (count)
/* count is r3 */                /* eei: enable extern interrupts */
  wrteei 0                       /* Stop interrupts if enabled */
  mtdec          r3              /* Move to DEC register */
  mtdecar        r3              /* Load same initial value to DECAR */
  lis            r0, 0x0440      /* Enable DEC interrupt and auto-reload */
                                 /* 0000 0100 0100 0000
                                  *   DIE = 1  decrementer interrupt enable
                                  *   ARE = 1  auto-reload enable */
  mttcr          r0
  li             r0, 0x4000      /* Enable Time Base and Decrementer */
  mthid0         r0
  lis            r4, dec_isr@h
  ori            r4, r4, dec_isr@l
  mtivor10       r4             /* IVOR10 contains interrupt vector for DEC */
}
```

- Routine enables interrupts and writes a count value to DECAR register (assembly code - more about this later)
- `init_DEC(count)` called by
  `init_interrupts(void (*fctn_ptr)(), int freq)`
- Example: Call your ISR by invoking
  `init_interrupts(isrB, 1000); /* Run isrB at 1000 Hz */`

isrA: Read Duty Cycle from Signal Generator

- See lab assignment for details
- ISR frequency: 20 kHz
- Sine wave: 1 kHz, 1 to 4 volts, external input
- PWM:
  - 20 kHz and 60 kHz frequency (DIP selectable)
  - Duty cycle proportional to voltage input
- Procedure:
  - Turn on LED 0.
  - Read AN0 analog input
  - Calculate duty cycle
  - Set the PWM duty cycle
  - Turn off LED 0.

isrB: Calculate Duty Cycle from sin()

- See lab assignment for details
- ISR frequency: 1 kHz
- Sine wave: 100 Hz, numerically calculated by sin() function
- PWM: 60 kHz, 40% to 60% duty cycle
- Procedure:
  - Turn on LED 0.
  - Calculate sin(2*pi * i / 10), i.e., 10 times per period, hence *i* is incremented by 1 each invocation.
  - Set the PWM duty cycle
  - Turn off LED 0.

isrC: Calculate Duty Cycle Table Look-up

- See lab assignment for details
- Essentially the same as isrB, except pre-calculate sin() and store as a look-up table
- What's the advantage?