ECE 381 - Microcontrollers

Lecture 3 - Interrupts

SOUTHERN ILLINOIS UNIVERSITY
EDWARDSVILLE

Last Class...

- For Lab 2, we introduced the concept of polling
  - Check for change in bit
  - If changed, do routine, else check again
- Polling is inefficient, can cause you to miss events, and is generally a bad idea
- Wouldn't it be much better if some background process or hardware told you when your event happened so you could do other work in the mean time?

Interrupts Are The Answer

- Interrupts handle asynchronous events
  - A change in a bit on a GPIO pin
  - An internal timer going off
  - A new ADC sample
- Interrupts signal the CPU (IRQ – Interrupts Request) to run an associated Interrupt Service Routing (ISR)
- The Interrupt Vector Table has a hard coded address specific to each type of interrupt so the CPU knows where to find the ISR

Interrupt Control

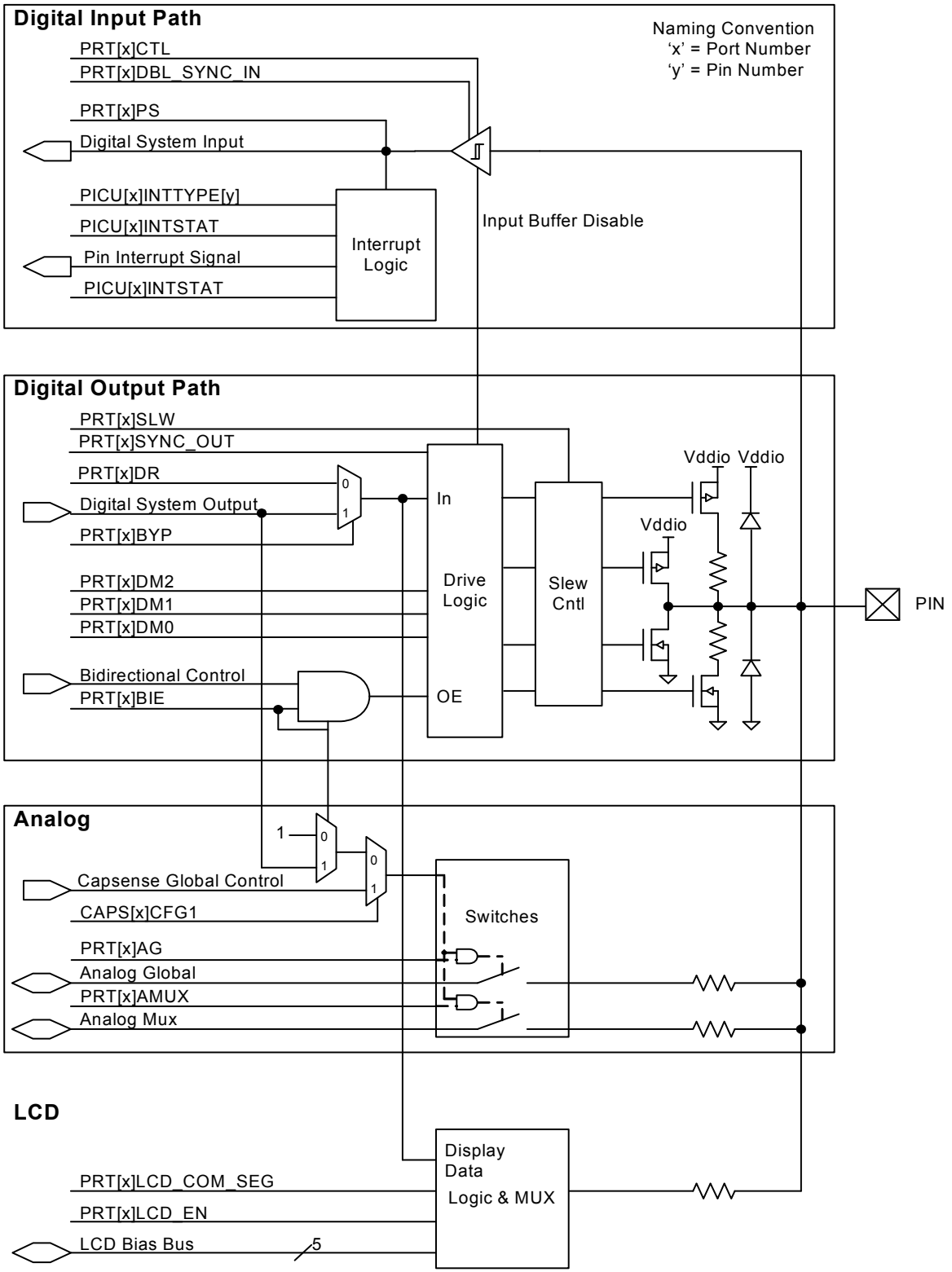
- Global Interrupt Enable – A bit that must be set to allow any interrupt (PsoC5: *M8C_EnableGInt*)
- Interrupt Mask – A bit specific to each type of interrupt which must be set to allow that specific interrupt (Some interrupts are nonmaskable!)
- Interrupt Flag – Another bit that signals which interrupt is active. Typically must be cleared before another interrupt of that type can signal.

Typical Interrupt Driven Program

- Interrupt setup
  - Enable Interrupts you are using (set flags, masks, etc.)
  - Set interrupt mode (high, low, level change, etc.)
  - Set up ISR if necessary
- Rest of code
- Enable Global Interrupt
- Infinite Loop (or Wait for Interrupt instruction)
  - Low Power: Go to sleep

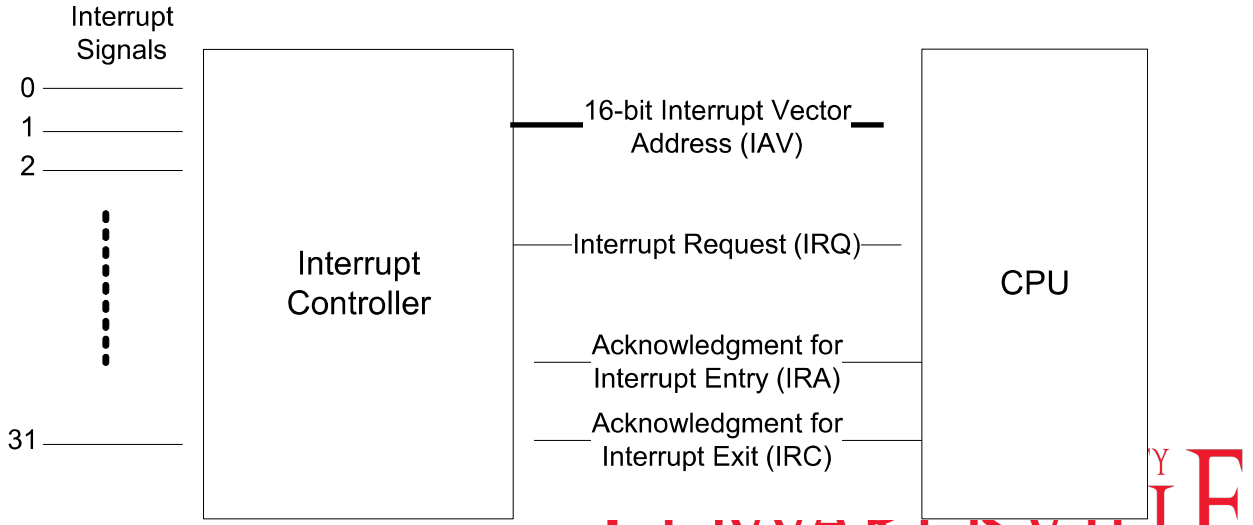
Interrupt Service Routines

- When the CPU gets an interrupt, it has to stop current operations and run the associated ISR
- This means it has to save the current program state (PC, SP, Accumulators(?), Status Regs., etc.)
  - These typically get pushed onto the stack (which resides in RAM)
  - May also disable other interrupts
  - Takes time (interrupt latency)
- Run ISR (Keep it short and sweet!)
  - Sometimes just set a flag → main infinite loop checks for flag → runs code on interrupt
- Return From Interrupt Instruction at end of ISR
  - Different from subroutine because of flags! (#pragma interrupt_handler for PSoC)
  - Typically pulls program state off the stack for you (otherwise you have to code it yourself)


Digital Input Path
Naming Convention
'x' = Port Number
'y' = Pin Number
PRT[x]CTL
PRT[x]DBL_SYNC_IN
PRT[x]PS
Digital System Input
PICU[x]INTTYPE[y]
PICU[x]INTSTAT
Pin Interrupt Signal
PICU[x]INTSTAT
Interrupt Logic
Input Buffer Disable
Digital Output Path
PRT[x]SLW
PRT[x]SYNC_OUT
PRT[x]DR
Digital System Output
PRT[x]BYP
PRT[x]DM2
PRT[x]DM1
PRT[x]DM0
Bidirectional Control
PRT[x]BIE
In
Drive Logic
OE
Slew Cntl
Vddio
Vddio
Vddio
PIN
Analog
1
Capsense Global Control
CAPS[x]CFG1
PRT[x]AG
Analog Global
PRT[x]AMUX
Analog Mux
Switches
LCD
PRT[x]LCD_COM_SEG
PRT[x]LCD_EN
LCD Bias Bus
5
Display Data
Logic & MUX

PSoC5 Interrupt Architecture

- 32 Interrupt Lines (8 Priority Levels)
  - Each line has associated vector address for ISR
  - 0 is highest priority (7 lowest)
- Programmable Vector Addresses
  - Reduce Interrupt latency (CPU can directly branch)
- Flexible Interrupt Sources
  - No predefined interrupt types (ie. GPIO, Timer, etc.)
  - Generic Interrupt user module for any source



Assignments

- Lab 3 – Interrupts and Quadrature Encoders
- Chapter 7 in the Technical Reference Manual
- Cypress Application Note AN54460
- Chapter 10 in Cady