Programming for Embedded Systems
Lecture 3: Digital to Analog Conversion

Bernhard Firner

February 9, 2015

RUTGERS

Digital to Analog Conversion?

- The MSP430 can only output two values: on and off
- How can we create an analog signal (e.g. sin wave)?
- The answer is Digital to Analog Conversion (DAC)
- We can do a specific kind of DAC with the MSP430 - Pulse Width Modulation (PWM)

DAC with PWM

1. Divide our analog signal into time slices
2. Turn our digital signal on for a fraction of the time slice to match the total energy output of the digital signal for that time slice
3. Use reconstructions filter to smooth the digital output into an analog signal
    - A reconstruction filter is basically an integrating RC circuit/low pass filter

Applications

- Signal generation
- Decoding of digitally encoded voice (cell phones, voip)
- Music players

Sampling Rate

- The *sampling rate* is the number of digital samples used to reconstruct the analog signal per unit time
- The higher the sampling rate the better the reconstruction
- However, faster sampling rates will require more processing speed

Sample Resolution

- In pulse width modultion we create a pulse that lasts for a fraction of the sampling period
- The number of different durations available is the *resolution* or *depth*
- For example:
  - Assume turning a pin on or off happens immediately
  - The MSP430 is running at 16MHz
  - We are reconstructing a signal with 4MHz sampling rate
  - Each sample lasts for four clock cycles
  - Our pulse is thus 0, 1, 2, 3, or 4 cycles long

The Effects of Rate and Resolution

- Low resolutions loose the actual signal
  - Imagine a song sampled every 5 seconds – could you tell what it was?
- Low depth reconstruction results in digital looking signals
  - Low bitrate music is like this – notes move in discrete steps

PWM in the MSP430

1. Set a pin to output mode
2. Set Timer A to "Up" mode (count to TACCR0)
3. Set TACCR0 and clock speed to determine sampling rate
   - We will call every count from 0 to TACCR0 a sample
4. When TAR resets to 0 set the pin high
5. Once TAR reaches the appropriate pulse width set pin low

MSP430 PWM Support

- The MSP430 can do steps 4 and 5 for us automatically
- Save a few lines of code and an interrupt handler
- TI's application note is posted on sakai
- Let's look at the family user's guide and chip manual

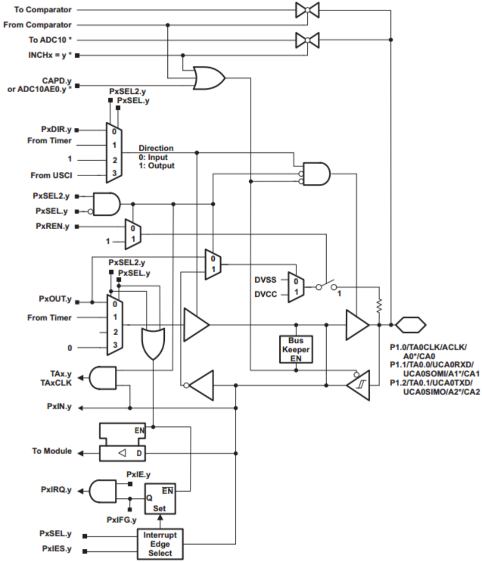
Family User's Guide

- PWM output is covered in section 12.2.5 (pgs 363-364)
- It is not covered well
- Output mode 7 will:
  - Turn off after reaching TACCRx
  - Turn back on after reach TACCR0
- In UP mode this means the pin is on from 0 to TACCRx and off otherwise

Which Pin?

- The user guide does not mention anywhere how to select a pin
- Only in the application note
- Can find this info in SLAS735 (MSP430G2x53 device guide)

Port 1 Schematic



- TA0.x mean TimerA 0
- TA0.0 is TA0CCR0
- TA0.1 is TA0CCR1

Timer Control

- Certain output pins can be timer controlled
- Set direction to output and set pin in PxSEL register

```
//Set P1.6 to TA0.1 control
P1DIR |= BIT6;
P1SEL |= BIT6;
```

RUTGERS

More on Output Select

- Each pin has different special outputs
- See the port schematics starting on pg72 of the G2x53 device guide
- We want TACCR0 to control the sampling rate
- We want TACCR1 to control the pulse widths
- Let's use P1.6 since it is also connected to the green LED

Example Code - Set Up

```
void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

    //Set the green LED to output
    P1DIR |= BIT6;
    //Set up output from Timer A0.1
    P1SEL |= BIT6;

//Make sure we are running with DCO at 16MHz
BCSCTL1 = CALBC1_16MHZ;
DCOCTL = CALDCO_16MHZ;

    //Set up the Timer A module
    //Set up TACCR0 to 1000
    //Sourced from a clock running at 16MHz this
    //will give us 16,000 samples/second
    TACCR0 = 1000;

    //TACCR1 will hold the bin width for each sample
    TACCR1 = 0;
```

Example Code - Control Registers

```
...
    //Set up the Timer A module
//UP mode, no division, SMCLK source
    TACTL = MC_1 | ID_0 | TASSEL_2;

    //Enable an interrupt when TAR == TACCR0
    TACCTL0 = CCIE;
    //Turn on internal PWM system that outputs on P1.6
    //Mode 7 will be on from 0 to TACCR1 and off until TACCR0
    TA0CCTL1 = OUTMOD_7;

    __enable_interrupt();
}
```

The Interrupt

```
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TACCR0_INT(void) {
//Set TACCR1 to the next sample value
//This interrupt is triggered once per sample
}
```

Annoying Details

- The value in TACCR1 needs to be updated in the interrupt
- On the next clock cycle, TAR reset to 0
- If TACCR1 is > 0 then the output pin goes high
- If we aren't dividing the clock, this gives only 1 cycle to update

Problems

- Having only one clock cycle causes problems
  - How can we compute a function, like sin?
  - How about loading data from external memory?
- The solution is buffering
  - We pre-calculate the next value after TAR resets
  - It will be immediately available to load later

Using a Mutex

- We will use a variable as a mutex

```
unsigned int buffer_empty = 0;
unsigned int buffer = 0;
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TACCR0_INT(void) {
//Set TACCR1 to the next sample value
//This interrupt is triggered once per sample
TACCR1 = buffer;
buffer_empty = 1;
}
```

In Main

- In main we update the buffer

```
while (1) {
    //Wait for buffer to be empty
    while (!buffer_empty);
    //Calculate the next pulse width value
    buffer = some complex calculation;
}
```

More Timing Concerns

- The MSP430G2553 has no hardware multiplier
- Floating point operations are incredibly slow
- For many functions it is simpler to use a *lookup table*

Lookup Tables

- A lookup table is just an array of values
- Let's say we want to output a sin wave with 100 samples
  - We record $sin(x * 2\pi/100)$ for x = 1 ... 100
  - In our main function we keep track of a sample index
  - Each time we increment the index and load that next value into the buffer
- This saves processing time at the cost of memory

Next Lab and Project

- Next class we'll have a lab
- You will use PWM to generate sawtooth and sin waves
- For the next project, you'll use sin waves to generate tones. You'll put several tones together to synthesize some simple digital music