Lab 2: Quadrature Decoding using the eTPU
Lab 2: Quadrature Decode

- Use “slow mode” quadrature decode
- Read the optical encoder and update a 16-bit position count register to track wheel position
  - in counts and
  - as angular position
- Use the debugger to verify wheel position
- Output position to 16 LEDs and demonstrate overflow and underflow
Lab 2: eTPU

- Time Processing Unit (TPU) is a co-processor designed for timing control.
- TPU operates in parallel with the CPU.
- Built-in functions or user-programmable out of dedicated RAM.
Lab 2: eTPU

- Freescale provides special purpose eTPU software for many different functions
  - AC and DC motor control
  - Automotive applications including crankshaft position sensing, spark and fuel control
  - Quadrature decode
    - MPC5553 has built-in quadrature decode on a different peripheral device (eMIOS) – but we’ll use the eTPU
Typical eTPU Example

Ignition control can be accomplished without CPU intervention
Files and Documents

• Reference material you will want to read:
  – Freescale Application Note AN2842: Using the Quadrature Decoder (QD) eTPU Function
    • Operating modes, performance
    • Application programming examples: initialization, value return functions
    • Section 18.4 Memory Map/Register Definition
Files and Documents

• Freescale files that you will have to include in your code:
  - `etpu_set.h` /* Auto-generated etpu code */
  - `etpu_util.h` /* Function prototypes */
  - `etpu_util.c` /* Functions */
  - `etpu_qd.h` /* fqd function prototypes */
  - `etpu_ppa.h` /* Pulse and period accumulation function prototypes */
Files and Documents

• You are given \texttt{fqd.h}, function prototype header file
• You need to write the functions in \texttt{fqd.c}
• You are given a template file \texttt{fqd\_template.c}
  \begin{itemize}
  \item \texttt{init\_FQD();} /* initialize the eTPU */
  \item \texttt{ReadFQD\_pc();} /* read encoder position */
  \item \texttt{updateCounter();} /* update wheel position */
  \item \texttt{updateAngle();} /* convert counts to angle */
  \end{itemize}
• Also need to write
  \begin{itemize}
  \item \texttt{Lab2.c} /* read the encoder position, update position count and output the result to the LED */
  \item \texttt{Lab2angle.c} /* convert count to angle */
  \end{itemize}
Notes on Casting

• We need to read the position count register and accumulate a running count of wheel position:

\[ \text{NEW\_TOTAL} = \text{LAST\_TOTAL} + (\text{CURR\_FQDPC} - \text{PREV\_FQDPC}); \]

• \text{NEW\_TOTAL} and \text{LAST\_TOTAL} are signed 32-bit integers

• \text{CURR\_FQDPC} and \text{PREV\_FQDPC} are unsigned 16-bit integers*

• Will this code work?

* Count register is really 24 bits … we’ll use only the lower 16 bits to make life difficult and demonstrate a point
Notes on Casting

• **Recall integral promotion:**
  – Before basic operation ( + - * / ), both operands converted to same type
  – The smaller type is “promoted” to the larger type
  – *Value of promoted type is preserved*

• **Suppose**
  – LAST TOTAL = 0x00007FFF
  – CURR FQDPC = 0xFFFF
  – PREV FQDPC = 0x0000

• CURR FQDPC - PREV FQDPC = 0xFFFF

• CURR FQDPC - PREV FQDPC **promoted to 32-bit signed integer** = 0x00000000

• Wrong! Large positive value, not one step negative
Notes on Casing

• Do this:

\[ \text{NEW\_TOTL} = \text{LAST\_TOTAL} + (\text{int16\_t})(\text{CURR\_FQDPC} - \text{PREV\_FQDPC}); \]

• First cast \text{CURR\_FQPC} and \text{PREV\_FQPC} as 16-bit signed integers

• The result will be sign-extended and summed with the 32-bit signed value, \text{LAST\_TOTAL}

\[ 0x0007FFF + 0xFFFFFFFF = 0x0007FFE \]

-1 (base 10) The correct answer: 1 step backwards