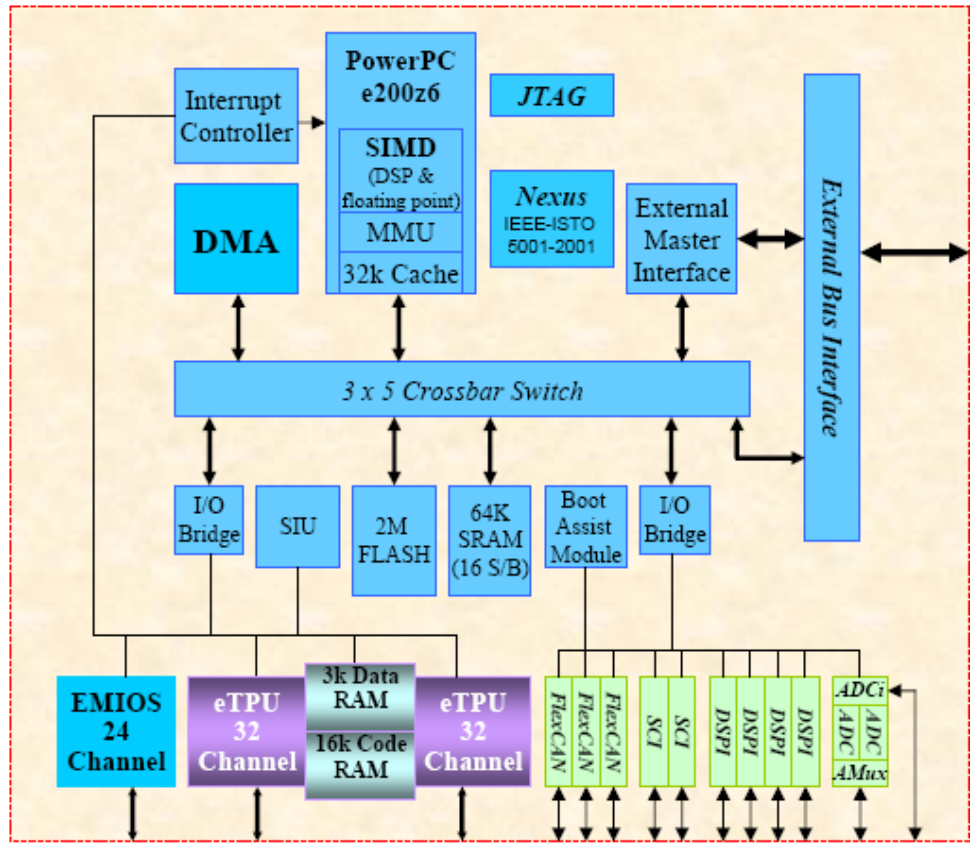
Lab 2: Quadrature Decoding using the eTPU

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

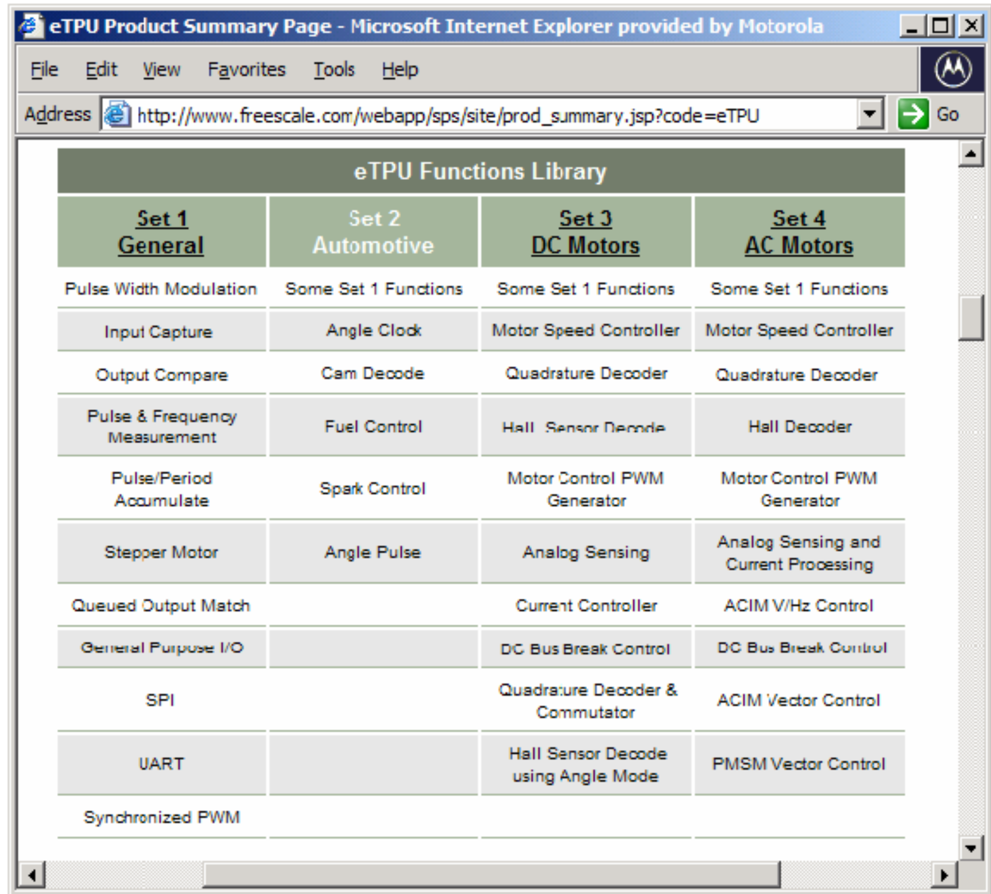
Lab 2: eTPU

- 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

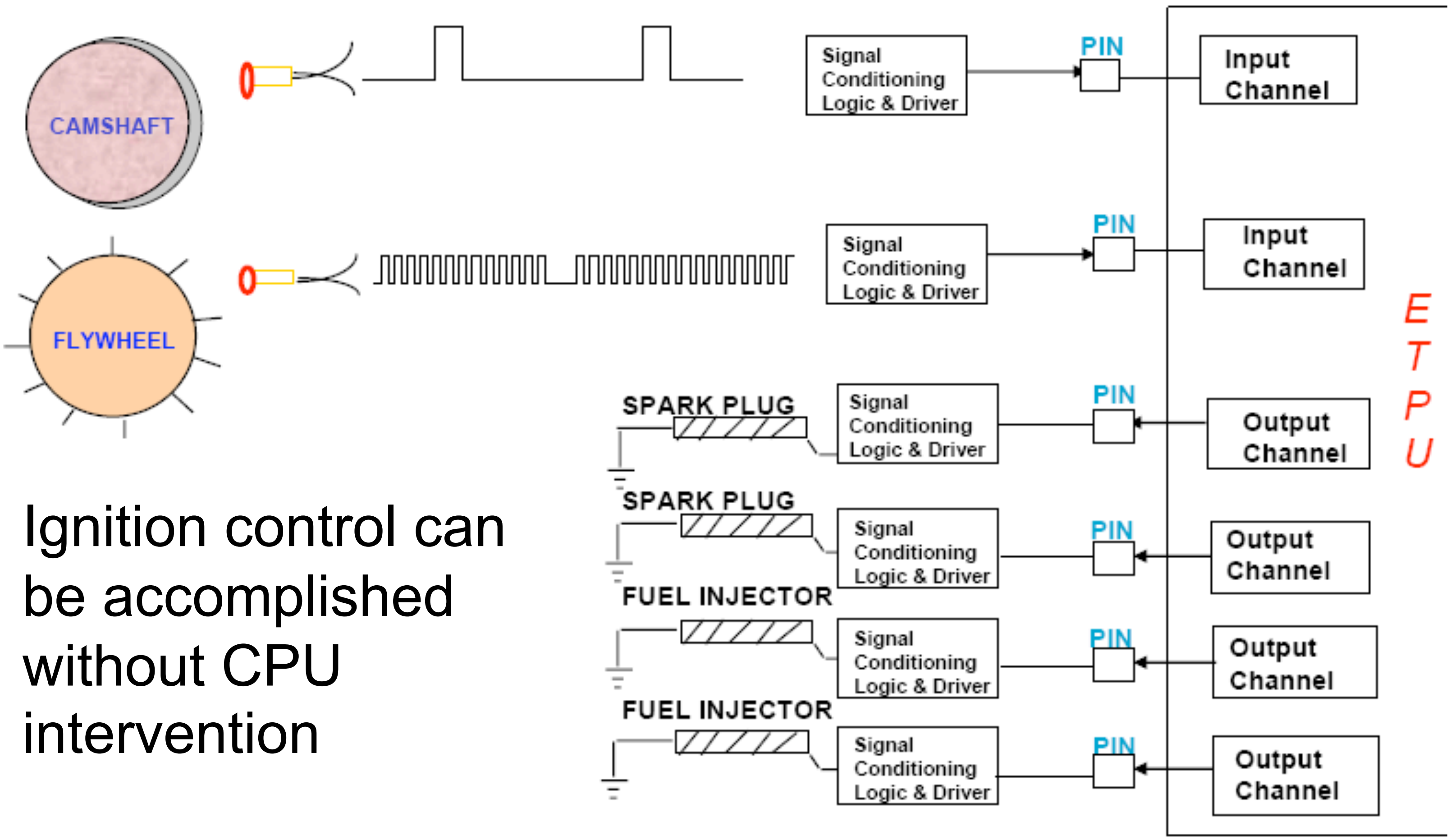
- Freescale provides special purpose eTPU software for many different functions
  - AC and DC motor control
  - Automotive applications including crankshaft position sensing, spark and fuel control
  - Quadrature decode
    - MPC5553 has built-in quadrature decode on a different peripheral device (eMIOS)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



eTPU Functions Library

Set 1 General	Set 2 Automotive	Set 3 DC Motors	Set 4 AC Motors
Pulse Width Modulation	Some Set 1 Functions	Some Set 1 Functions	Some Set 1 Functions
Input Capture	Angle Clock	Motor Speed Controller	Motor Speed Controller
Output Compare	Cam Decode	Quadrature Decoder	Quadrature Decoder
Pulse & Frequency Measurement	Fuel Control	Hall Sensor Decode	Hall Decoder
Pulse/Period Accumulate	Spark Control	Motor Control PWM Generator	Motor Control PWM Generator
Stepper Motor	Angle Pulse	Analog Sensing	Analog Sensing and Current Processing
Queued Output Match		Current Controller	ACIM V/Hz Control
General Purpose I/O		DC Bus Break Control	DC Bus Break Control
SPI		Quadrature Decoder & Commutator	ACIM Vector Control
UART		Hall Sensor Decode using Angle Mode	PMSM Vector Control
Synchronized PWM			

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
  - MPC5553 Microcontroller Reference Manual
    - 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 `fqd.h`, function prototype header file
- You need to write the functions in `fqd.c`
- You are given a template file `fqd_template.c`
  - `init_FQD();        /* initialize the eTPU */`
  - `ReadFQD_pc();      /* read encoder position */`
  - `updateCounter();   /* update wheel position */`
  - `updateAngle();     /* convert counts to angle */`
- Also need to write
  - `Lab2.c    /* read the encoder position, update position count and output the result to the LED */`
  - `Lab2angle.c  /* convert count to angle */`

Notes on Casting

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

```
NEW_TOTAL = LAST_TOTAL + (CURR_FQDPC - PREV_FQDPC);
```

  - `NEW_TOTAL` and `LAST_TOTAL` are signed 32-bit integers
  - `CURR_FQDPC` and `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`

  1 step backwards
- `CURR FQDPC - PREV FQDPC = 0xFFFF`
- `CURR FQDPC - PREV FQDPC` promoted to 32-bit signed integer = `0x0000FFFF`
- Wrong! Large positive value, not one step negative

Notes on Casing

- Do this:

```
NEW_TOTL = LAST_TOTAL + (int16_t)(CURR_FQDPC - PREV_FQDPC);
```

- First cast `CURR_FQPC` and `PREV_FQPC` as 16-bit signed integers
- The result will be sign-extended and summed with the 32-bit signed value, `LAST_TOTAL`

```
0x0007FFF + 0xFFFFFFFF = 0x0007FFE
```

-1 (base 10)

The correct answer:
1 step backwards