Lab 2: Quadrature Decoding using the FlexTimer

Fall 2019
Lab 2: Quadrature Decode

- Use quadrature decode function of the S32K144 FlexTimer module
- Read the optical encoder and update a 16-bit position count register to track wheel position
  - in counts and
  - as angular position
- Output position to 16 LEDs and demonstrate overflow and underflow
Lab 2: Quadrature Decode

- The haptic wheel in the lab uses a 1000 CPR (cycles per revolution) encoder from Avago.
- For an explanation of this terminology, see the website usdigital.com/support/resources/glossary

- For a general explanation of encoders, see the website ni.com/tutorial/7109/en
FlexTimer Module (FTM)

- S32K144 has four FlexTimer modules, FTM0-3.
- Each FTM has 8 channels that may be used for input capture, output compare, or PWM generation (Lab 4).
- Two FlexTimer modules, FTM1-2, may be used for quadrature decoding – we will use FTM2 for QD.
- Each FTM has a 16-bit counter that, when used for quadrature decoding, is incremented or decremented depending on encoder input.
FlexTimer accepts several sources of inputs, including channels 0-7, and primary and secondary quadrature signals, referred to as phase A and phase B.

We will use pin A13 for the primary quadrature signal (phase A) and pin A12 for the secondary quadrature signal (phase B).
Configuring the PCRs for QD

- In Lab 1, you configured pins for general purpose I/O by writing appropriate values to pin configuration registers (PCRs).
- To configure the FlexTimer for quadrature decoding, you must program the PCRs associated with pins A12 and A13 for inputs from quadrature decoding signals.
- To find the addresses of these PCRs, see Section 12.1.2.
- From the table in the spreadsheet S32K144_IO_Signal_Description_Input_Multiplexing/IO Signal Table we see that the MUX bitfield should be set to \(0b110\).
FTM Registers and Memory Map

• Several registers are used to configure each FTM – we will only need a few.

45.4.3.1 FTM Memory map

FTM0 base address: 4003_8000h
FTM1 base address: 4003_9000h
FTM2 base address: 4003_A000h
FTM3 base address: 4002_6000h

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Width (In bits)</th>
<th>Access</th>
<th>Reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0h</td>
<td>Status And Control (SC)</td>
<td>32</td>
<td>RW</td>
<td>0000_0000h</td>
</tr>
<tr>
<td>4h</td>
<td>Counter (CNT)</td>
<td>32</td>
<td>RW</td>
<td>0000_0000h</td>
</tr>
<tr>
<td>8h</td>
<td>Modulo (MOD)</td>
<td>32</td>
<td>RW</td>
<td>0000_0000h</td>
</tr>
<tr>
<td>Ch</td>
<td>Channel (n) Status And Control (C0SC)</td>
<td>32</td>
<td>RW</td>
<td>0000_0000h</td>
</tr>
<tr>
<td>10h</td>
<td>Channel (n) Value (C0V)</td>
<td>32</td>
<td>RW</td>
<td>0000_0000h</td>
</tr>
<tr>
<td>14h</td>
<td>Channel (n) Status And Control (C1SC)</td>
<td>32</td>
<td>RW</td>
<td>0000_0000h</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Counter Register: \texttt{CNT}

- The \texttt{CNT} register contains the current value of the 16-bit quadrature decode counter. We will need to read the value of this counter and compare it to the previous value to determine the change in wheel position.
Counter Initial Value (CNTIN) and Modulo (MOD) Registers

The CNTIN register contains the initial value INIT of the 16-bit quadrature decode counter. This is the value the counter will return to after it “rolls over” by counting up to the value set in the MOD register.

- Set INIT = 0x0000 and MOD = 0xFFFF to use all 16 bits of the counter.
Features Mode Selection (MODE) Register

- The MODE register contains several bitfields used to configure the FTM.
- We shall only need to ensure that the Write Protection Disable is set so that we can write to write-protected registers (WPDIS=1). (Note that 1 is the reset value of this bitfield.)

### Diagram

![Diagram of the MODE register showing bitfields and their default values.]
The FlexTimer has optional filters for the inputs from the quadrature signals. These are useful to suppress noise bursts that might be interpreted as valid rising or falling edges of the signals. We will disable these filters by setting $\text{CH2FVAL} = 0x0$, its reset value. The effect of the noise filters on noise immunity and latency will be explored in the homework.
The SC register is used to configure the FTM.
The FLTPS bitfield is used to set the value of an additional prescaler that affects the operation of the FlexTimer filter. We do not use the filter and thus leave FLTPS = 0x0, its reset value (prescaler = 1).
In Lab 4, we will set several bitfields in the SC register to use the FTM for PWM signal generation.
The QDCTRL register has several bitfields that configure the FTM for quadrature decoding.

- All these should be left at their reset values of 0 except the Quadrature Decoder Mode Enable bit, which should be set to enable quadrature decoding (QUADEN=1).
- If set, the bits PHAFLTREN and PHBFLTREN enable input filters on the two quadrature signal. We leave these bits at their reset value of 0.
- Note: QUADMODE=0 selects 4X decoding as described in lecture.
/** FTM - Register Layout Typedef */
typedef struct {
  __IO uint32_t SC;       /**< Status And Control, offset: 0x0 */
  __IO uint32_t CNT;      /**< Counter, offset: 0x4 */
  __IO uint32_t MOD;      /**< Modulo, offset: 0x8 */
  ...                      /**< Features Mode Selection, offset: 0x54 */
  __IO uint32_t SYNC;     /**< Synchronization, offset: 0x58 */
  ...                      /**< Input Capture Filter Control, offset: 0x78 */
  __IO uint32_t FLTCTRL;  /**< Fault Control, offset: 0x7C */
  __IO uint32_t QDCTRL;   /**< Quadrature Decoder Control And Status, offset: 0x80*/
  ...                      /**< Quadrature Decoder Control And Status, offset: 0x80*/
} FTM_Type, *FTM_MemMapPtr;

/** Peripheral FTM2 base address */
#define FTM2_BASE (0x4003A000u)
/** Peripheral FTM2 base pointer */
#define FTM2 ((FTM_Type *)FTM2_BASE)

/* QDCTRL Bit Fields */
#define FTM_QDCTRL_QUADEN_MASK 0x1u
#define FTM_QDCTRL_QUADEN_SHIFT 0u
#define FTM_QDCTRL_QUADEN_WIDTH 1u
#define FTM_QDCTRL_QUADEN(x) /* see S32K144.h */
You are given `qd.h`, function prototype header file

You need to write three functions in `qd.c`

You are given a template file `qd_template.c`
- `Init_QD();` /* initialize the FlexTimer */
- `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 */
Notes on Casting

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

\[
\text{curr\_count} = \text{FTM2}\rightarrow\text{CNT} \& \text{FTM\_CNT\_COUNT\_MASK}; \\
\text{total\_count} = \text{total\_count} + (\text{curr\_count} - \text{last\_count});
\]

• \text{total\_count} \text{ is a signed 32-bit integer}
• \text{curr\_count} and \text{last\_count} \text{ are unsigned 16-bit integers}
• Will this code work?
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**
  - `total_count = 0x00007FFF`
  - `curr_count = 0xFFFF`
  - `last_count = 0x0000`

  - `curr_count - last_count = 0x0000FFFF - 0x00000000 = 0x0000FFFF`

- Wrong! Large positive value, not one step negative

- "-" operand promotes `curr_count` and `last_count` to 32 bit signed integers:
Notes on Casting

- Do this:
  \[
  \text{total\_count} = (\text{total\_count} + (\text{int16\_t})(\text{curr\_count}-\text{last\_count}));
  \]

- Cast \((\text{curr\_count}-\text{last\_count})\) to a 16-bit signed integer

- The result will be sign-extended and summed with the 32-bit signed value, TOTAL

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

The correct answer: 1 step backwards
S32K144 Clocks

- The S32K144 system clock can be set to
  - 112 MHz (HSRUN mode)
  - 80 MHz (RUN mode)
- Various other clock frequencies set in `eecs461.h`:
  ```c
define SPLL_CLK_FREQ 160000000u /* 160 MHz */
define SYS_CLK_FREQ 80000000u /* 80 MHz */
define BUS_CLK_FREQ 40000000u /* 40 MHz */
define SPLLDIV2_CLK_FREQ 40000000u /* 40 MHz */
define SPLLLDIV1_CLK_FREQ 10000000u /* 10 MHz */
define SOSC_CLK_FREQ 80000000u /* 8 MHz */
define SOSCDIV1_CLK_FREQ 80000000u /* 8 MHz */
define SOSCDIV2_CLK_FREQ 80000000u /* 8 MHz */
```
- In Lab 2 we need to know that the FlexTimer clock is set to
  `SPLLLDIV1 = 10 MHz`
- See Chapters 27-29