Lab 3: Queued A-D Conversion (eQADC)
Queued Analog-to-Digital Conversion

• Acquire analog input from the potentiometer and observe the result using the debugger

• Using an oscilloscope, measure the time required to complete one conversion by toggling GPIO

• Acquire a sine wave signal from the function generator and investigate aliasing
  – Generate a square wave signal from the input sine function and observe output signal frequency on the digital oscilloscope
  – Use the “software oscilloscope” to output the acquired signal to the serial port for display on the monitor
Queued Analog-to-Digital Conversion

- Chapter 19 MPC5553-RM
  - Two 12-bit ADC (ADC0/1)
  - Single ended, 0-5v
  - Double ended -2.5 – 2.5v
  - 40 MUXed input channels
- Command FIFO (CFIFO) triggers ADC
- Results FIFO (RFIFO) receives conversions
- DMA (Direct Memory Access) transfers
  - Commands from user-defined command queue to CFIFO
  - Results from RFIFO to user-defined results queue
Queued Analog-to-Digital Conversion

- 6 CFIFOs (EQADC_CFIFO[0-5])
- 6 RFIFOs (EQADC_RFIFO[0-5])
- Any CFIFO can command either ADC, and results can be sent to any RFIFO
  - We will configure EQADC_CFIFO0 and EQACD_CFIFO1 to use ADC0 and put the results in RFIFO0 and RFIFO1 respectively
  - Write commands to CFIFO “push registers” and read results from RFIFO “pop registers”
Queued Analog-to-Digital Conversion

• **Operating Modes**
  – *Single-scan mode*
    • Command Queue is scanned one time
    • Software involvement is needed to re-arm queue after queue is scanned
  – *Continuous-scan mode*
    • Command Queue is scanned multiple times
    • Software involvement is not needed to re-arm queue

• **All modes may be software-triggered, edge-triggered or level-triggered**
  – We will set up 2 queues:
    • EQADC_CFIFO0 for software-triggered single scan
    • EQADC_CFIFO1 for software-triggered continuous scan
Programming the eQADC

- Like other peripherals, the eQADC must be configured by writing commands to special purpose registers
  - eQADC Module Configuration Register (EQADC MCR)
  - CFIFO Control Registers (EQADC CFCRn)
- Structure to access these registers is included in \texttt{MPC5553.h}
  - EQADC\_MCR described in Section 19.3.2.1 of the Reference Manual
  - EQADC\_CFCRn described in Section 19.3.2.6 and Tables 19-9 and 19-10
**EQADC_MCR**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ESSIE: Synchronous Serial Interface enable (disable = 00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DBG: Debug mode enable (disable = 00)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of EQADC_MCR](image)

Figure 19-2. eQADC Module Configuration Register (EQADC_MCR)
EQADC_CFCRn

- **SSE**: Single scan enable
- **CFINV**: CFCR invalidate (CFINV = 0)
- **MODE**: Operating mode (Table 9-10)
  - 0000: disabled
  - 0001: software triggered single scan
  - 1001: software triggered continuous scan
Programming the eQADC

• Unlike other peripherals, some eQADC registers are *not* accessible to the programmer

• Registers that control on-chip ADCs are programmed by sending 32-bit *configuration* and *command* messages to the CFIFO
  
  – Write Configuration Command Message
    • Sets the control registers of the on-chip ADCs.
  
  – Read Configuration Command Message
    • Reads the contents of the on-chip ADC registers which are only accessible via command messages
  
  – Conversion Command Message
    • Conversion result is returned with optional time stamp
Non-memory Mapped ADC Registers

- 5 configuration registers for each ADC
- Control register (ADC\textsubscript{n} CR)
  - Enables ADC
  - Enables external multiplexing
  - Sets the ADC clock speed
- Other configuration registers enable time stamp and set calibration parameters
ADCn_CR

- **ADCn_EN**: Enable ADC
- **ADCn EMUX**: Enable MUX
- **ADCn_CLK_PS**: ADC clock prescaler (see Table 19-28)
**R/W Configuration Command Message Format**

- **EOQ**: end-of-queue
- **PAUSE**: wait for trigger
- **EB**: external buffer (0 for on-chip ADC)
- **BN**: buffer number (0 or 1)
- **R/W**: 0 = write; 1 = read command message
- **ADC_REGISTER HIGH BYTE**: value to be written into the most significant 8 bits of control/configuration register when the R/W bit is negated
- **ADC_REGISTER LOW BYTE**: value to be written into the least significant 8 bits of control/configuration register when the R/W bit is negated
- **ADC_REG_ADDRESS**: ADC register address (see Tables 19-25 and 26)

See Tables 19.35 and 36
Configuration Command Message Format

- As usual, we can use a structure or union to construct a configuration command message
- Access as a register or individual bit fields

```c
union adc_config_msg
{
    vuint32_t R;
    struct
    {
        vuint32_t header:6;
        vuint32_t command:26;
    } BB;
    struct
    {
        vuint32_t EOQ:1;
        vuint32_t PAUSE:1;
        vuint32_t :3;
        vuint32_t EB:1;
        vuint32_t BN:1;
        vuint32_t RW:1;
        vuint32_t HIGH_BYTE:8;
        vuint32_t LOW_BYTE:8;
        vuint32_t ADC_REG_ADDRESS:8;
    } B;
};
```
Configuration Command Message
Format

Example: Select ACD0 and configure the ADC0_CR

union adc_config_msg config;
/* ADC configurations- command internal ADC register parameters
config.R = 0;
config.B.BN = 0;    /* Command ADC 0
config.B.HIGH_BYTE = 0x80;   /* Enable ADC module
config.B.LOW_BYTE = 0x05; /* Use clock prescaler of 10
config.B.ADC_REG_ADDRESS = 0x01;   /* ADC0_CR address
Conversion Command Message Format

- **EOQ** – End of Queue
  Set to ‘1’ to indicate last entry in Command Queue

- **PAUSE** – Enter Pause state after transfer of current command message
  (Queue waits for the next trigger state)

- **EB** – External Buffer
  Should be set to logic ‘0’ for the internal buffers.

- **BN** – Buffer Number
  0 = Message will be sent to Buffer #0
  1 = Message will be sent to Buffer #1

- **CAL** – Calibration Control
  Set to ‘1’ if conversion result needs to be calibrated.

- **MESSAGE_TAG** – RFIFO #
  Specifies which one of the six RFIFO the result is sent to.

- **LST** – Long Sample Time select
  A sample time of 2, 8, 64 or 128 ADC clock can be selected.

- **FMT** – Conversion Data Format
  0 = Right Justified
  1 = Right Justified Signed

- **CHANNEL_NUMBER**
  Selects the Analog Input Channel

- **Bits 24:31**
  Zero: word is a conversion command
  Nonzero: word is a configuration command

---

**Diagram:**
- **CFIFO Header**
- **ADC Command**
- **Time Stamp**
Conversion Command Message Format

• Our conversion command structure

union cfifo_msg{
    vuint32_t R;
    struct{
        vuint32_t header:6;
        vuint32_t command:26;
    } BB;
    struct{
        vuint32_t EOQ:1;
        vuint32_t PAUSE:1;
        vuint32_t :3;
        vuint32_t EB:1;
        vuint32_t BN:1;
        vuint32_t CAL:1;
        vuint32_t MESSAGE_TAG:4;
        vuint32_t LST:2;
        vuint32_t TSR:1;
        vuint32_t FMT:1;
        vuint32_t CHANNEL_NUMBER:8;
        vuint32_t :8;
    } B;
};
Conversion Command Message Format

Conversion command message example:

```c
union cfifo_msg cmd;
    cmd.R = 0;
    cmd.B.EOQ = 1; /* end-of-queue */
    cmd.B.PAUSE = 0;
    cmd.B.EB = 0;
    cmd.B.BN = 0; /* use first QADC unit */
    cmd.B.CAL = 0; /* no calibration */
    cmd.B.MESSAGE_TAG = 0b0000; /* result queue 0 */
    cmd.B.LST = 0b10; /* sample time = 2 clks */
    cmd.B.TSR = 0;
    cmd.B.FMT = 0;
    cmd.B.CHANNEL_NUMBER = single_channel;
```
Conversion Result Format

- 12-bit conversion is stored in a 16-bit RFIFO as a signed or unsigned integer
- In either case, the result is stored in bits [2:13], i.e., bit shifted 2 left
Direct Memory Access (DMA)

• Recall
  – We need to transfer ADC configuration and conversion commands from memory to CFIFO
  – We need to transfer results from the RFIFO to memory

• This could take lots of time if CPU intervention is required
Direct Memory Access (DMA)

DMA Controller
64 sources (MPC5554)
64 priority levels or round robin or mixture of both

12 eQADC
8 DSPI
15 eMIOS
9 eTPU A
4 External
12 eTPU B
4 eSCI
Direct Memory Access (DMA)

• DMA services:
  – peripheral requests (eQADC, for example)
  – software initiated requests

• Transfer Control Descriptor (TCD) used to define each channel (source and destination address, address increments, size etc..)

• See Chapter 9 in the Reference Manual, and Lab 3 document for description of DMA programming

• We have written the DMA code for Lab 3!
Function `setupDMARequests` continuously fills the CFIFO with commands from `CONT_SCAN_QUEUE` to read each ADC channel sequentially, and stores the results in `CONT_SCAN_RESULTS`.
Lab 3 Software

• As usual, you are given `qadc.h` with function prototypes; you will write the functions in `qadc.c`, plus application code in `lab3.c`
• Four functions (plus DMA) are required:
  - `qadcInit`: Initialize the eQADC:
    • Configure the conversion command queues
    • Configure the ADC
  - `fillCCMTable`: Build command conversion lists
    • Single and continuous scan lists required
  - `qadcRead1` and `qadcRead2`: Read the results
Lab 3 Software

- \texttt{qadcInit}: configuring the conversion command queues
  - Use the structure found in \texttt{MPC553.h} to access the MCR, CFIFO control registers and CFIFO push registers
  - Clear the MCR and set up one queue for software-triggered single scan and one queue for software-triggered continuous scan
  - Use the configuration command message structure to enable ADC0 and set the clock prescaler
Lab 3 Software

- **fillCCMTable**: Build conversion command lists
  - Use the conversion command message structure to build a single scan conversion command on `single_channel`
  - Use the conversion command message structure to build a queue of continuous scan command messages, `CONT_SCAN_QUEUE[x]`, where `x = channel_number`
    - Continuously scan ADC channels and put the results in the results queue
• **qadcRead1: Single scan**
  - Use the structure in MPC5553.h to
    - Write the command to the push register
    - Start scan (SSE = 1)
    - Wait until the scan is complete (wait until the results FIFO counter increments; see MPC5553-RM 19.3.2.8 “eQADC FIFO and Interrupt Status Registers,” RFCTR bits)
    - Read the results from the RFIFO pop register

• **qadcRead2: Continuous scan**
  - DMA is doing all the work: simply read the results from `CONT_SCAN_RESULTS` array
Lab 3 Assignment

• Basic Conversion Testing
  – Write a C program (lab3.c) that uses `qadcReadQ2` to retrieve the values of the eight analog inputs on the board and place them into an array named `iAnalogQ2`.
  – Use the debugger and slide potentiometer to verify that data are being acquired on each of the 8 input channels
  – Verify the `qadcReadQ1` function in a similar manner
Lab 3 Assignment

• **Timing**
  – Modify `lab3.c` so that, before the call to `qadcReadQ1` function, one of the LEDs is set to high and is set back to low after the function returns
  – Connect an oscilloscope to the GPO output pin and measure:
    • How long it takes for a scan to be completed
    • The periodic rate at which scans occur

• **Speed Testing**
  – Generate a square wave by toggling the GPIO with respect to an input signal threshold (a sine wave input will result in a square wave output of the same frequency).
  – Increase the input frequency and observe what happens
Lab 3 Assignment

- Oscilloscope Application
  - Build the software oscilloscope using the software provided
  - What is the highest-frequency signal you are able to capture and display without aliasing?