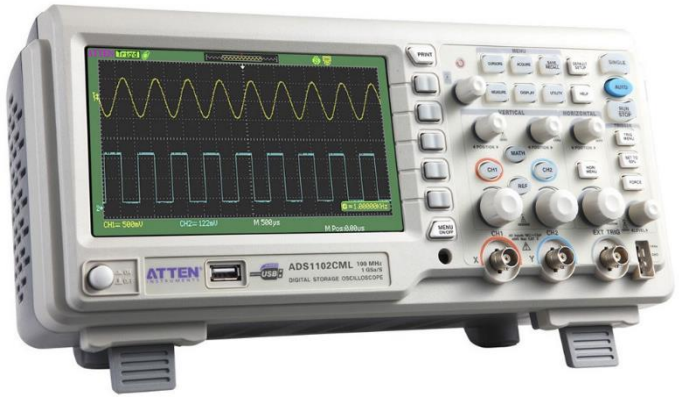
ECE 551
Project Spec

Fall ‘14



DSO with
Calibration
Capability

Grading Criteria: (Project is 30% of final grade)

- Project Grading Criteria:
  - Quantitative Element 25%

(yes this could result in extra credit)

$$Quantitative = \frac{EricSakshi\_ProjectArea}{YourSynthesizedArea}$$

Note: The design has to be functionally correct for this to apply

  - Project Demo (67.5%)
    - ✓ Code Review (12.5%)
    - ✓ Testbench Method/Completeness (12.5%)
    - ✓ Synthesis Script review (10%)
    - ✓ Post-synthesis Test run results (12.5%)
    - ✓ Results when placed in EricSakshi Testbench (20%)

Extra Credit Opportunity:

Appendix C of ModelSim tutorial instructs you how to run code coverage

- Run code coverage on a single test and get 1.5% extra credit
- Run code coverage across your test suite and get a cumulative number and get 3% extra credit.
- Run code coverage across your test suite and give concrete example of how you used the results to improve your test suite and get 4.5% extra credit.

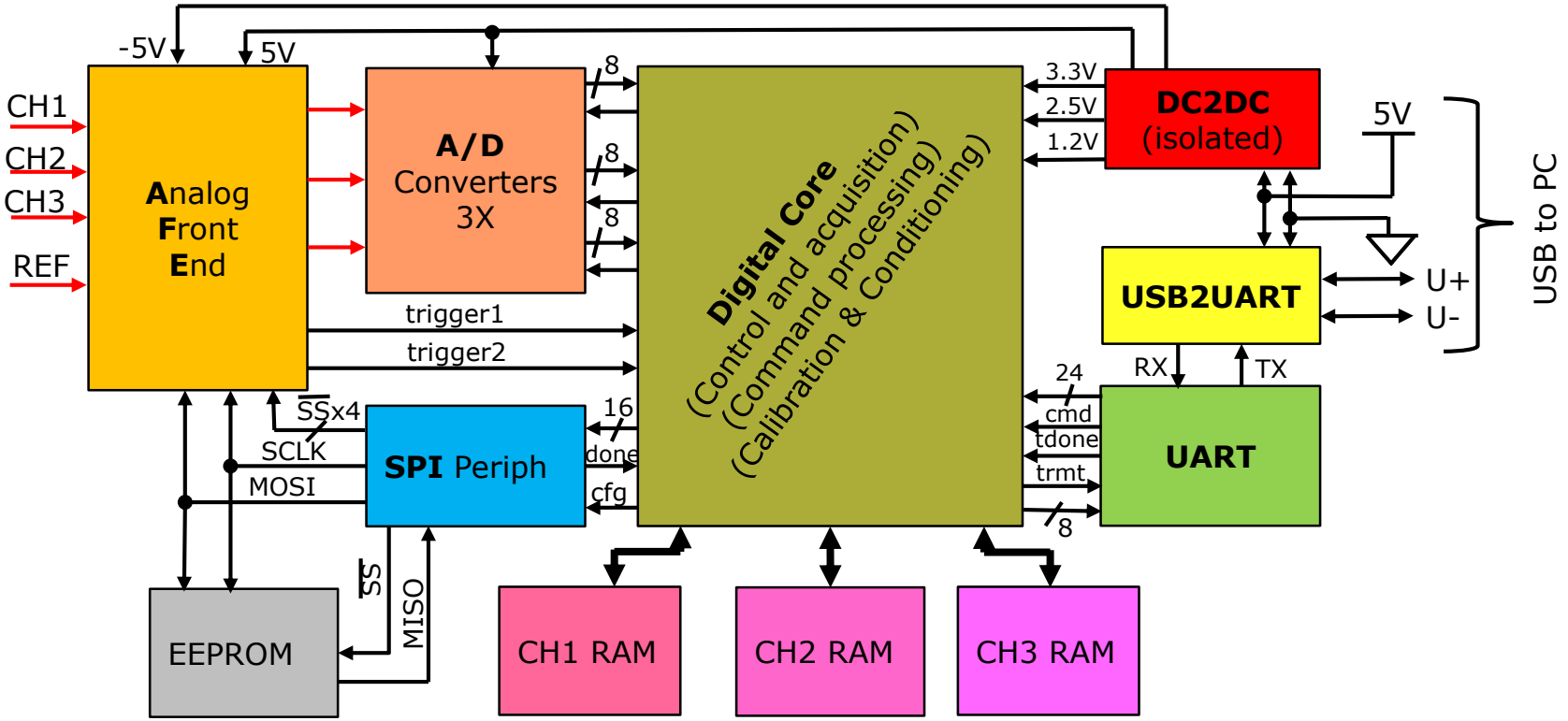
  - Project Report 7.5%
    - ✓ Expected to be around 3-6 pages (don't go crazy, I don't want to grade a really long report)
    - ✓ Show partitioning of digital core (block diagram, few words)
    - ✓ Report on datapath implementation (algorithm, support HW needed in datapath)
    - ✓ Report on each team members contributions
    - ✓ Section on lessons learned (focus on the hard leaned lessons…the heartache)

Project Due Date

- Project Demos will be held in B555:
  - Wednesday (12/10/14) from 1:00PM till evening.
  - Friday (12/12/14) from 1:00PM till evening.

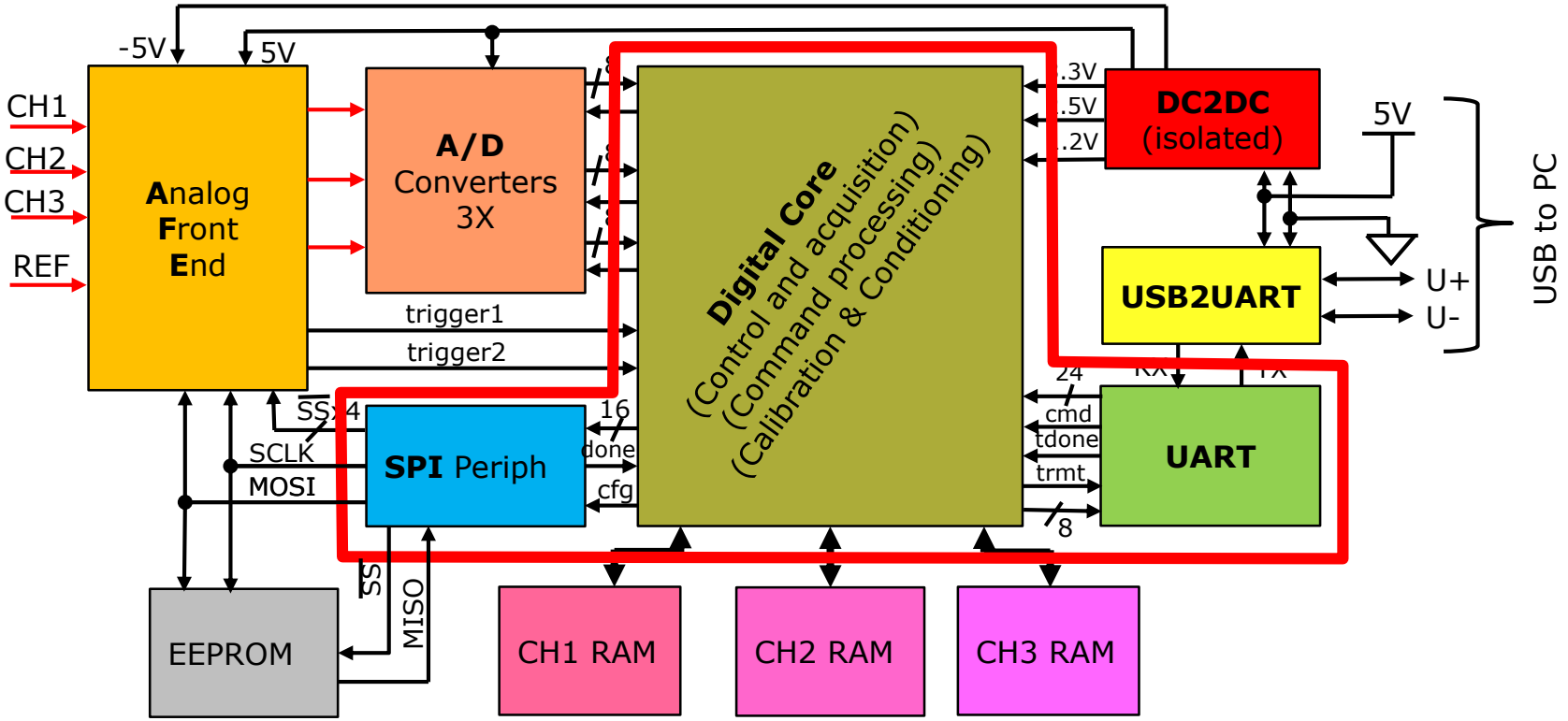
- Project Demo Involves:
  - ✓ Code Review
  - ✓ Testbench Method/Completeness
  - ✓ Synthesis Script & Results review
  - ✓ Post-synthesis Test run results
  - ✓ Results when placed in EricSakshi testbench
  - ✓ Hand in of short report

Block Diagram



This USB **DSO** contains a 3 channel AFE with independently configurable gain settings for each channel. The AFE also outputs two trigger sources (CH3 cannot be a trigger source). All AFE configuration (gains and trigger levels) are performed via a SPI interface. The analog outputs of the AFE go into 3 separate 20M sample/sec 8-bit A/D converters. A primary function of the digital core is to sample each A/D channel and store the result in the respective RAM bank. The digital core also performs the task of reading the raw samples, performing offset and gain correction, and sending the data to a USB port for display by an app running on a host PC. The host application also sets the AFE parameters by sending commands via USB. UART is an intermediate protocol for the host interface.

What is synthesized DUT vs modeled?

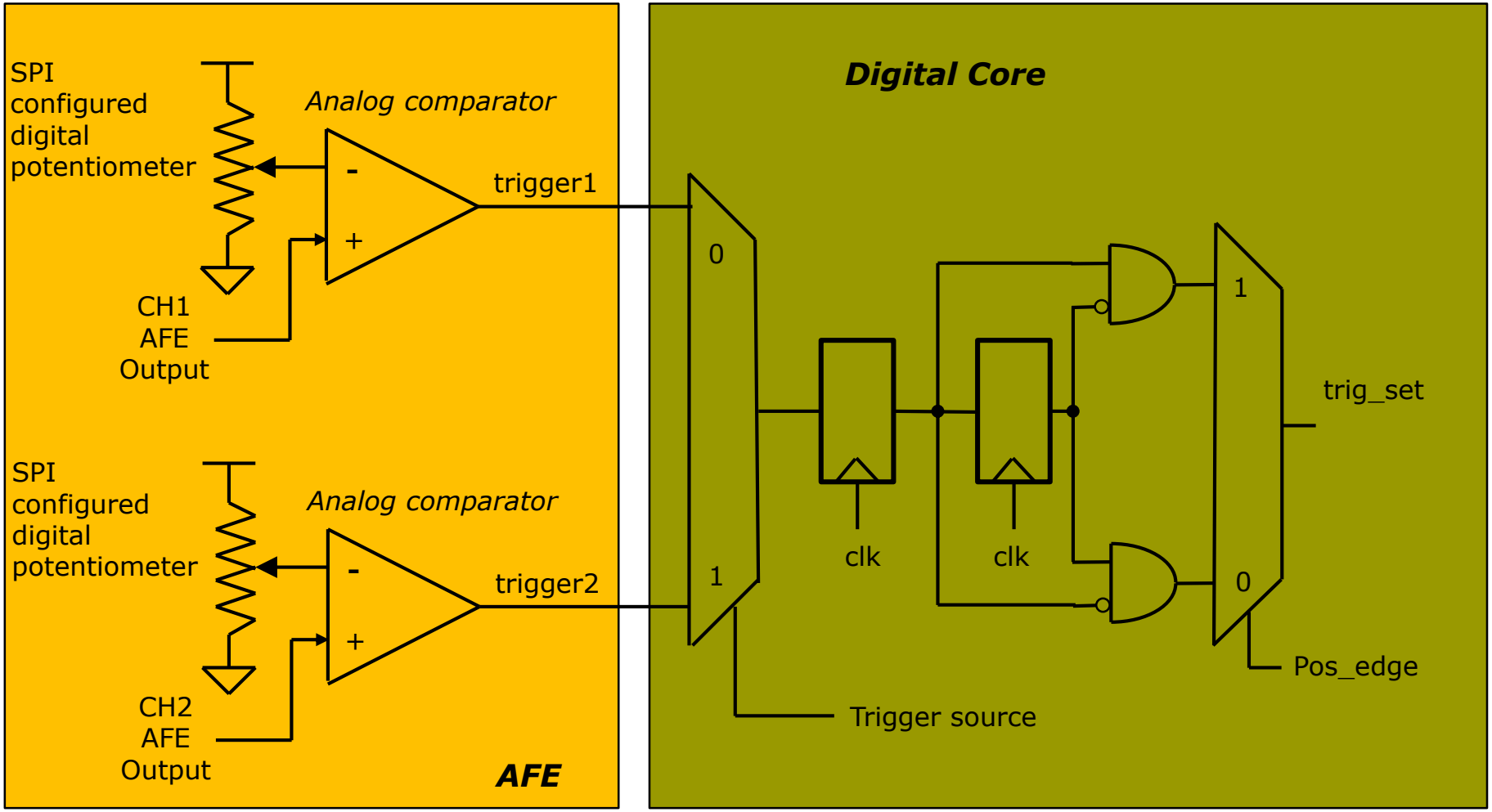


The blocks outlined in red above are pure digital blocks, and will be coded with the intent of being synthesized & APRed.

You Must have a block called **DSO_dig.v** which is top level of what will be the synthesized DUT.

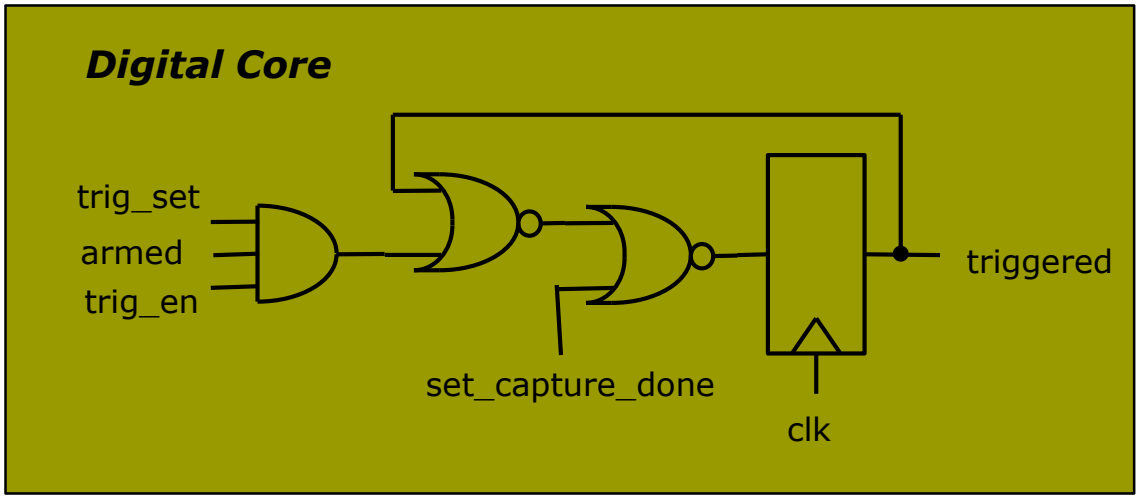
The 3 RAM blocks are not synthesized, but you will be creating a verilog model for these blocks. A verilog model of the A/D converter will be provided.

Trigger Logic:



You build the trigger logic to select CH1 vs CH2 and positive vs negative edge triggered. An output pulse on this trigger logic will set a S/R flop to indicate to the core that a trigger has occurred.

Trigger Logic (continued):



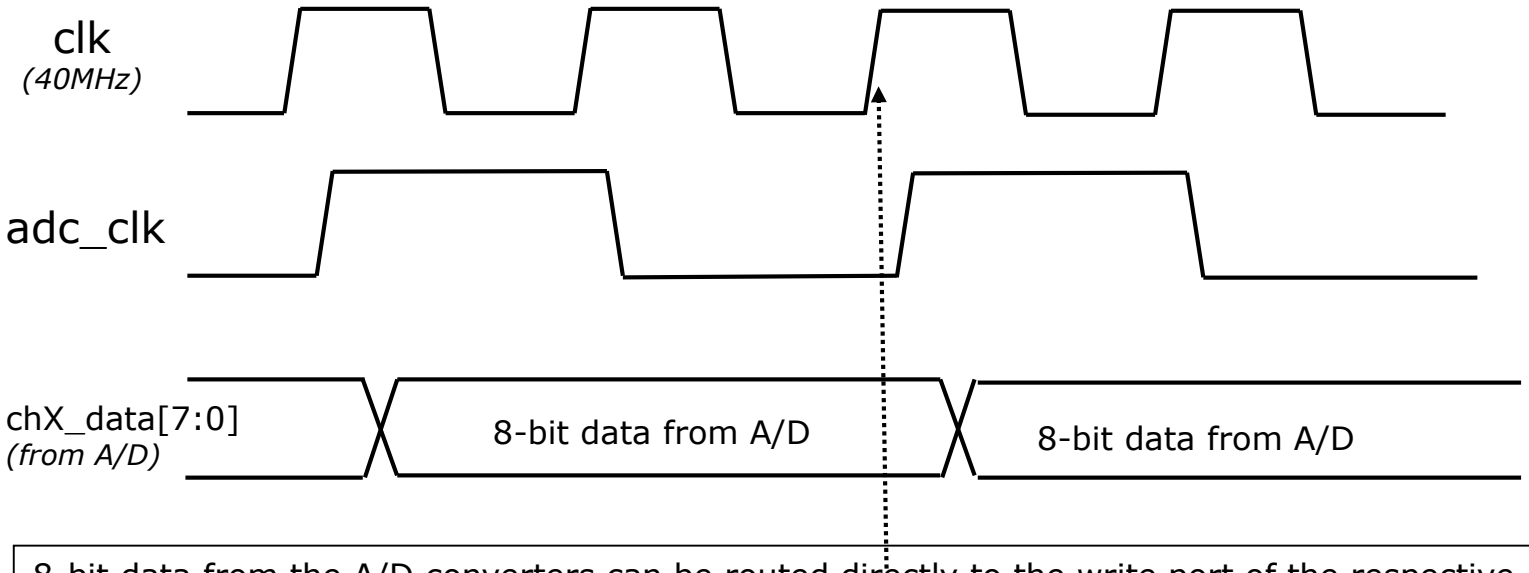
The trigger pulse (***trig_set***) from previous slide is only one clock cycle wide. This signal needs to be captured and preserved in an **S**et/**R**eset flop mechanism so a consistent signal (***triggered***) can be evaluated by the capture module.

The signal ***trig_en*** will be set when the DSO is in a run mode. When the host software has read all the channel data it will write to the ***trig_cfg*** register and clear the capture done bit.

The signal ***armed*** is asserted by the capture logic, and only allows a trigger to occur when the number of samples captured plus the number of samples to capture after trigger (***trig_pos***) will exceed the total sample memory size (512 entries).

A/D Interface:

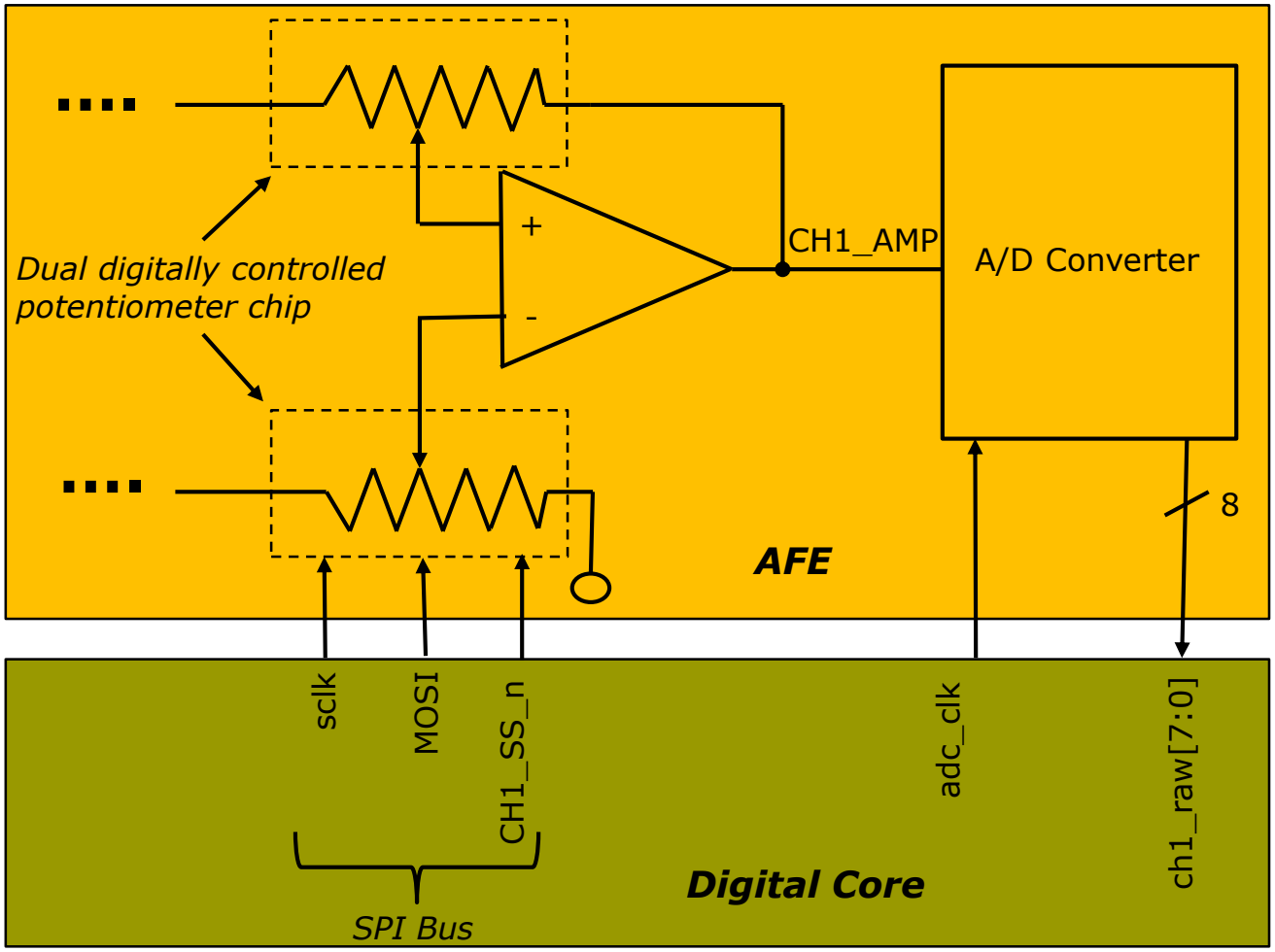
- The A/D interface is really simple. Feed the A/D converter a clock it will give you a new 8-bit sample on every positive edge of said clock (***adc_clk***).
- The system clock is 40MHz and the clock to the A/D converter is 20MHz and sourced from a simple 2:1 divider you implement in the digital core. We will just keep the A/D running the whole time. We are not too power constrained, so keep it simple.



8-bit data from the A/D converters can be routed directly to the write port of the respective RAM block. The ***adc_clk*** that drives the A/D converters should be opposite in phase of that used to drive the RAM blocks (RAM is also driven by 20MHz clock, but of opposite phase)

NOTE: Both ***adc_clk*** and ***rclk*** can be kept always running. No need to start/stop them.

AFE Configuration:



The digital core controls the gain of the channel amplifiers though the use of a dual digital potentiometer.

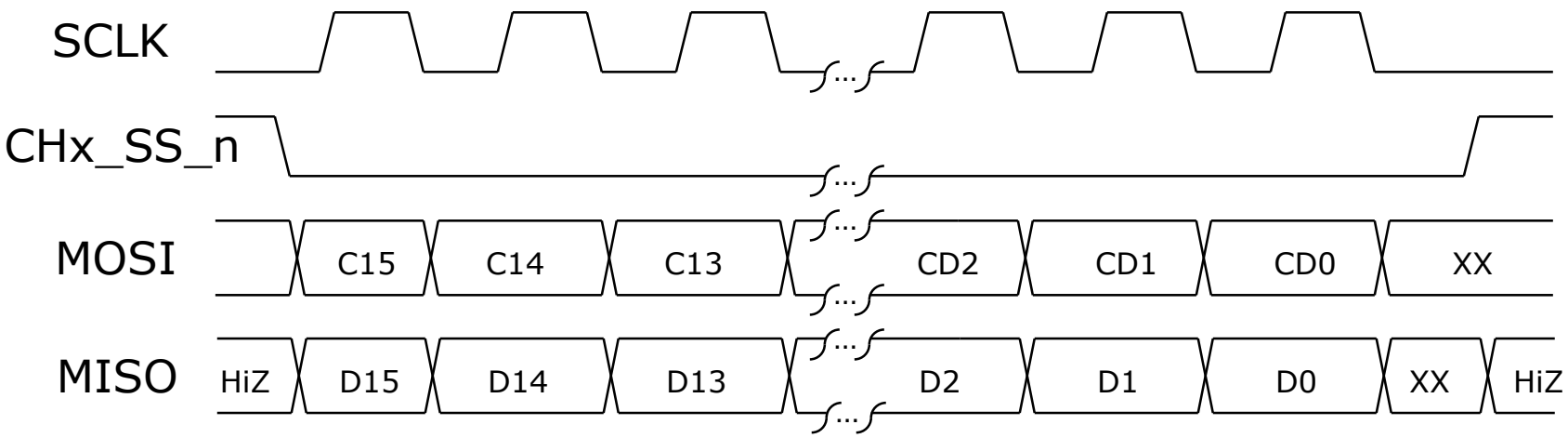
The digital potentiometers are set via a half-duplex SPI bus.

There are 4 such dual potentiometers in the AFE. One for each channel, and one to control the trigger levels for CH1 vs CH2 triggering.

What is SPI?

- Simple uni-directional serial interface (Motorola long long ago)
  - **S**erial **P**eripheral **I**nterconnect (very popular physical interface)
  - 4-wires for full duplex
    - ✓ MOSI (Master Out Slave In) (digital core will drive this to AFE)
    - ✓ MISO (Master In Slave Out) (not used in connection to AFE digital pots, only EEP)
    - ✓ SCLK (Serial Clock)
    - ✓ SS_n (Active low Slave Select) (Our system has 4 individual slave selects to address the 4 dual potentiometers, and a fifth to address the EEPROM)
  - There are many different variants
    - ✓ MOSI Sampled on clock low vs clock high
    - ✓ SCLK normally high vs normally low
    - ✓ Widths of packets can vary from application to applications
    - ✓ Really is a very loose standard (barely a standard at all)
  - We will stick to the most commonly used variant
    - ✓ MOSI changes on SCLK falling (is sampled by AFE on SCLK fall)
    - ✓ SCLK normally low
    - ✓ 16-bit packets

SPI Packets



A SPI packet inherently involves a send and receive (full duplex). The full duplex packet is always initiated by the master. Master controls SCLK, SS_n, and MOSI. The slave drives MISO if it is selected. If the slave is not selected it should leave MISO high impedance. The EEPROM is the only SPI peripheral that we have that will drive MISO. The digital potentiometers in the AFE are half-duplex and only received data.

MOSI will change on the falling edge of SCLK with the understanding that the slave will flop it on the falling edge of SCLK.

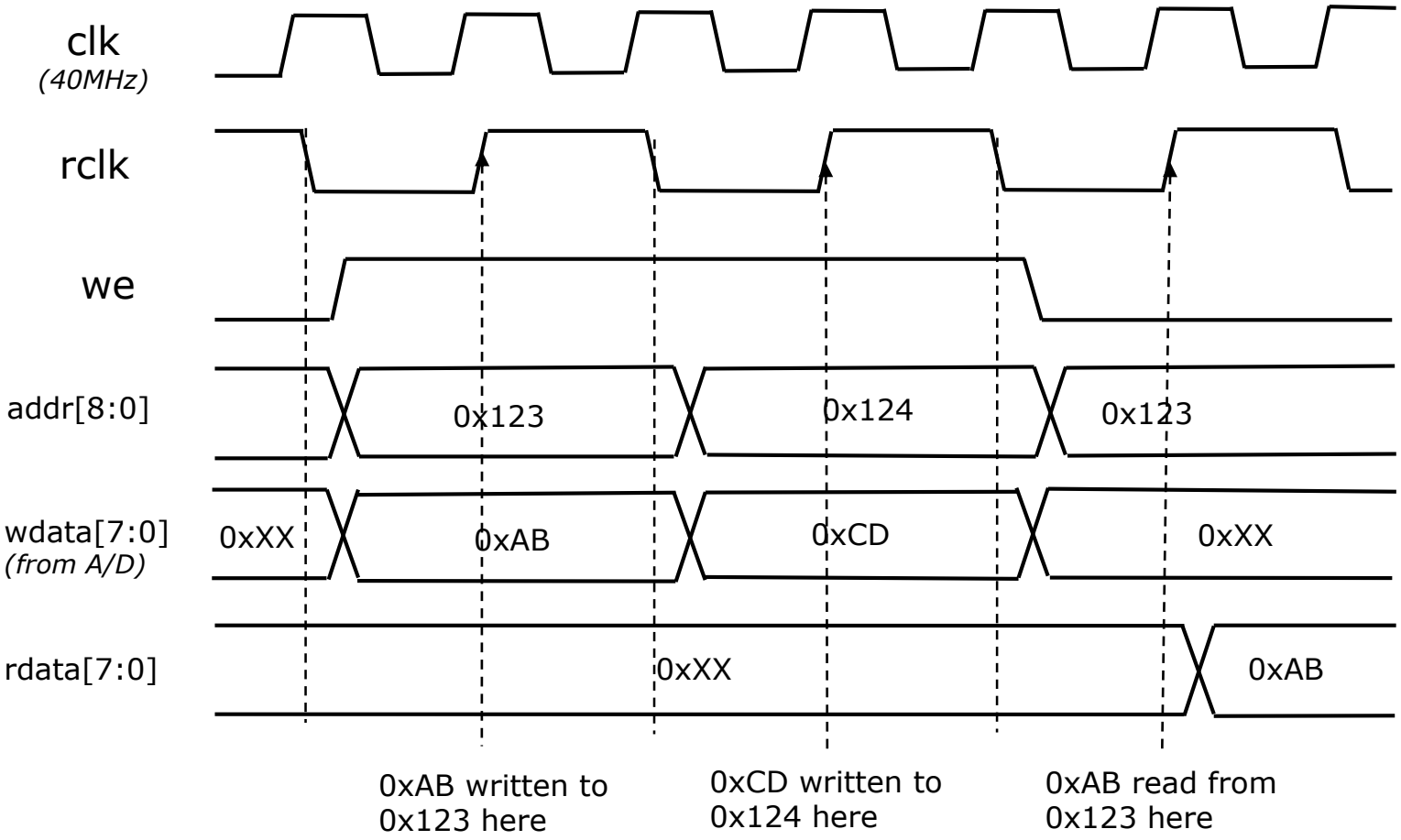
MISO will change on the falling edge of SCLK with the understanding that the master will flop it on the falling edge of the next SCLK.

RAM Blocks (store captured channel data)

There will be 3 dedicated RAM blocks to store the raw A2D data captured from each channel. Each of these RAM blocks are 8-bits in width with 512 entries. The RAM block are single ported, meaning there is one data port that can read or write to the RAM, but not both at the same time.

Signal:	Dir:	Description:
rclk	in	Clock to RAM block. Will be 20MHz (1/2 our system clock). All read/write operations occur on positive edge, therefore address and control should be setup on negative edge. Opposite in phase to ***adc_clk***
en	in	Active high enable. When high read/write operations are enabled. When low RAM is in low power inactive state
we	in	Active high write enable. Provided ***en*** is high, if ***we*** is also high the cycle will be a write, otherwise a read is occurring.
addr[8:0]	in	9-bit address to select one of 512 locations in RAM
wdata[7:0]	in	Write data. Drive this with the raw data from the A2D when capturing a trace
rdata[7:0]	out	Read data. When ***en*** is high and ***we*** is low then a read is being performed and the data read is available on this signal.

RAM Block Timing

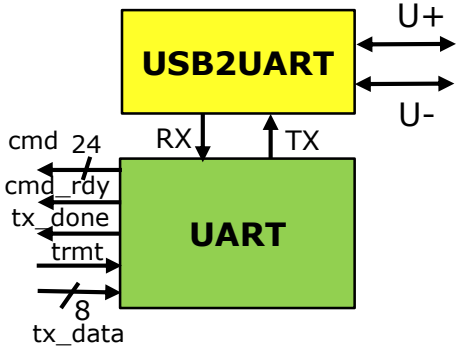


RAM read/write operation occur on the positive edge of ***rclk***. Given the timings for the A/D, *8-bit data* from each ADC can be routed direct to feed ***wdata*** to each RAM.

UART Interface

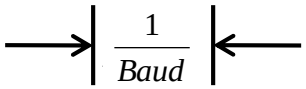
- UART serial interface will be used to communicate with host PC (through USB interface chip)
- Program running on host PC will configure settings of AFE
  - Gain for each channel (volts/div)
  - Trigger source, polarity, level
  - Timebase and sampling rate
- Program running on host PC will read and display the waveform data
  - Issues commands to read and transfer data from each RAM block to interface
- Program running on host PC will instruct scope to perform self calibration
  - Each channel can be calibrated individually for offset & span

Signal:	Dir:	Description
clk,rst_n	in	Clock and reset. 40MHz system clock, and active low reset
tx_data[7:0]	in	Data to transmit to host. Typically a positive acknowledge to a command, or a stream of bytes representing requested channel data
trmt	in	This signal is pulsed high for at least one clock cycle to start transmission
tx_done	out	Asserted when ***tx_data[7:0]*** has finished transmitting
cmd_rdy	out	Asserted when a new 24-bit command has been received from the host PC
cmd[23:0]	out	24-bit command
RX/ TX	in/ out	Serial data out (to USB chip, then to host PC) Serial data in (from USB chip, from host PC)

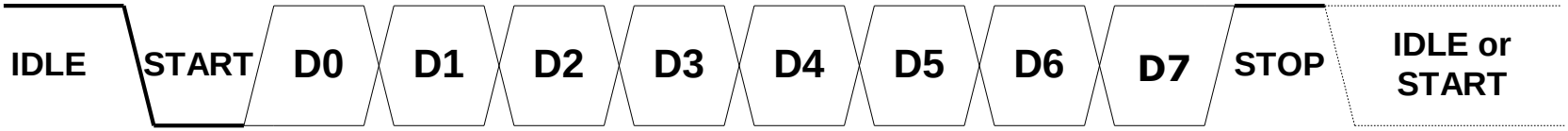


What is UART (RS-232)

- RS-232 signal phases
  - Idle
  - Start bit
  - Data (8-data for our project)
  - Parity (no parity for our project)
  - Stop bit – channel returns to idle condition
  - Idle or Start next frame



Baud rate will be 921,600 with 40MHz clock



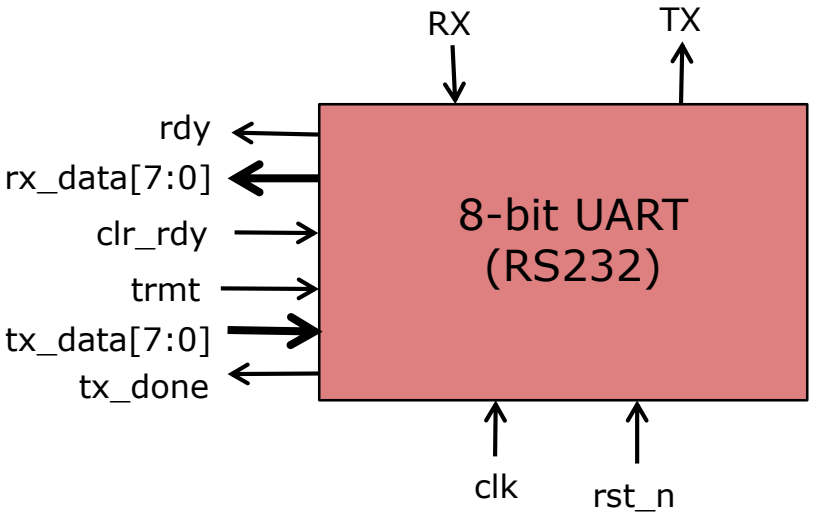
- Receiver monitors for falling edge of Start bit. Counts off 1.5 bit times and starts shifting (right shifting since LSB is first) data into a register.
- Transmitter sits idle till told to transmit. Then will shift out a 9-bit (start bit appended) register at the baud rate interval.

What is RS232 (UART)

- Full RS-232 involves handshaking signals
  - RTS, CTS, DTR, …
  - We will only use RX & TX assuming sender and receiver can "digest" bytes fast enough. Common use.

- When 8-bit data received UART will raise ***rdy*** and data will be presented on ***rx_data***[7:0].

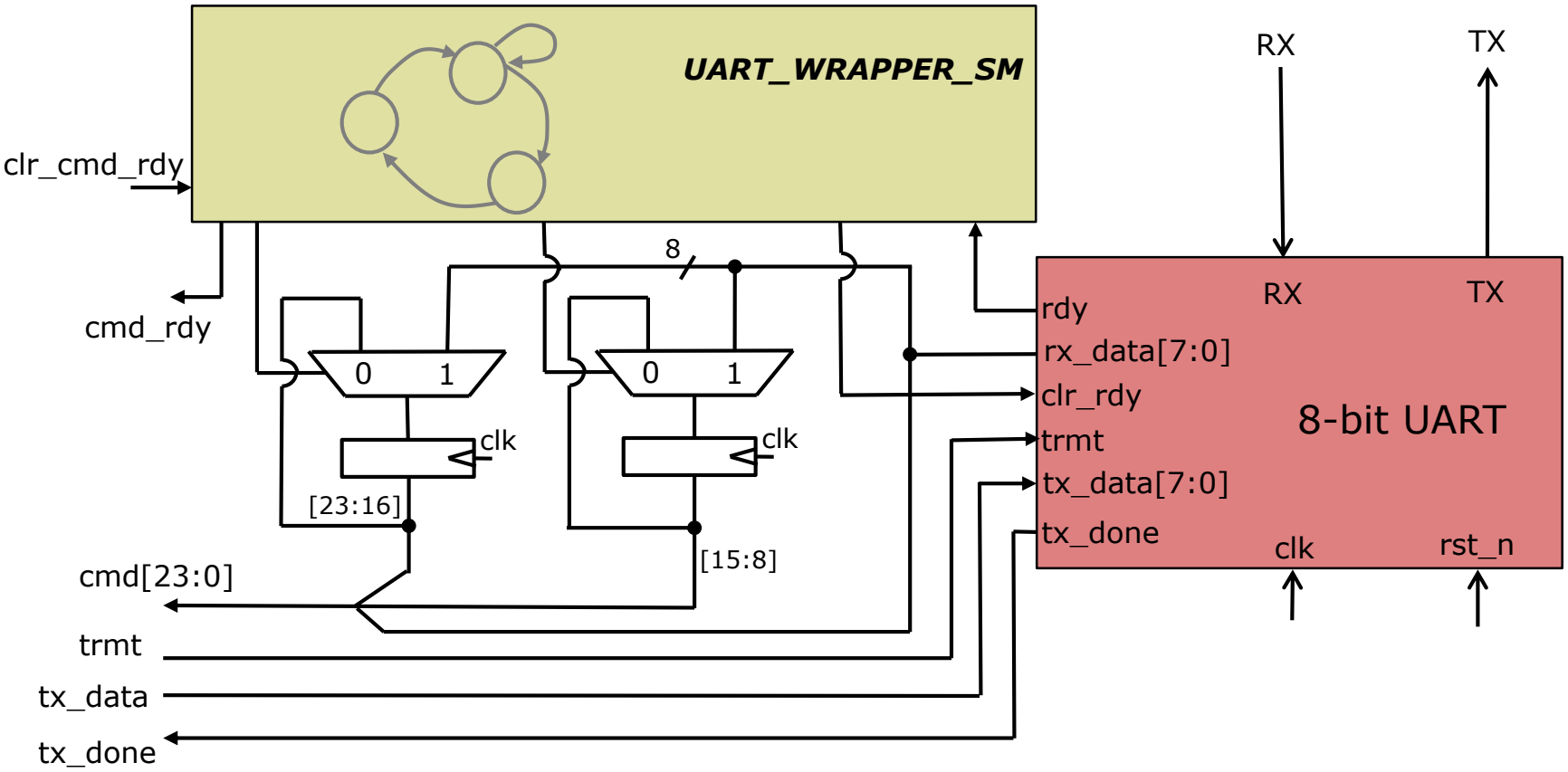
- When core has consumed data it will raise ***clr_rdy***, to indicate to the UART it should drop ***rdy***.



- To transmit the core will be present data on ***tx_data***[7:0] and then raise ***trmt***. When the UART is done transmitting it will raise ***tx_done***.

UART Wrapper

- Previous slides showed an 8-bit UART transceiver. We want to put a wrapper around it to make a unit that receives 24-bit command packets and sends 8-bit responses.



Digital Core Functionality (Analog Channel Capture)

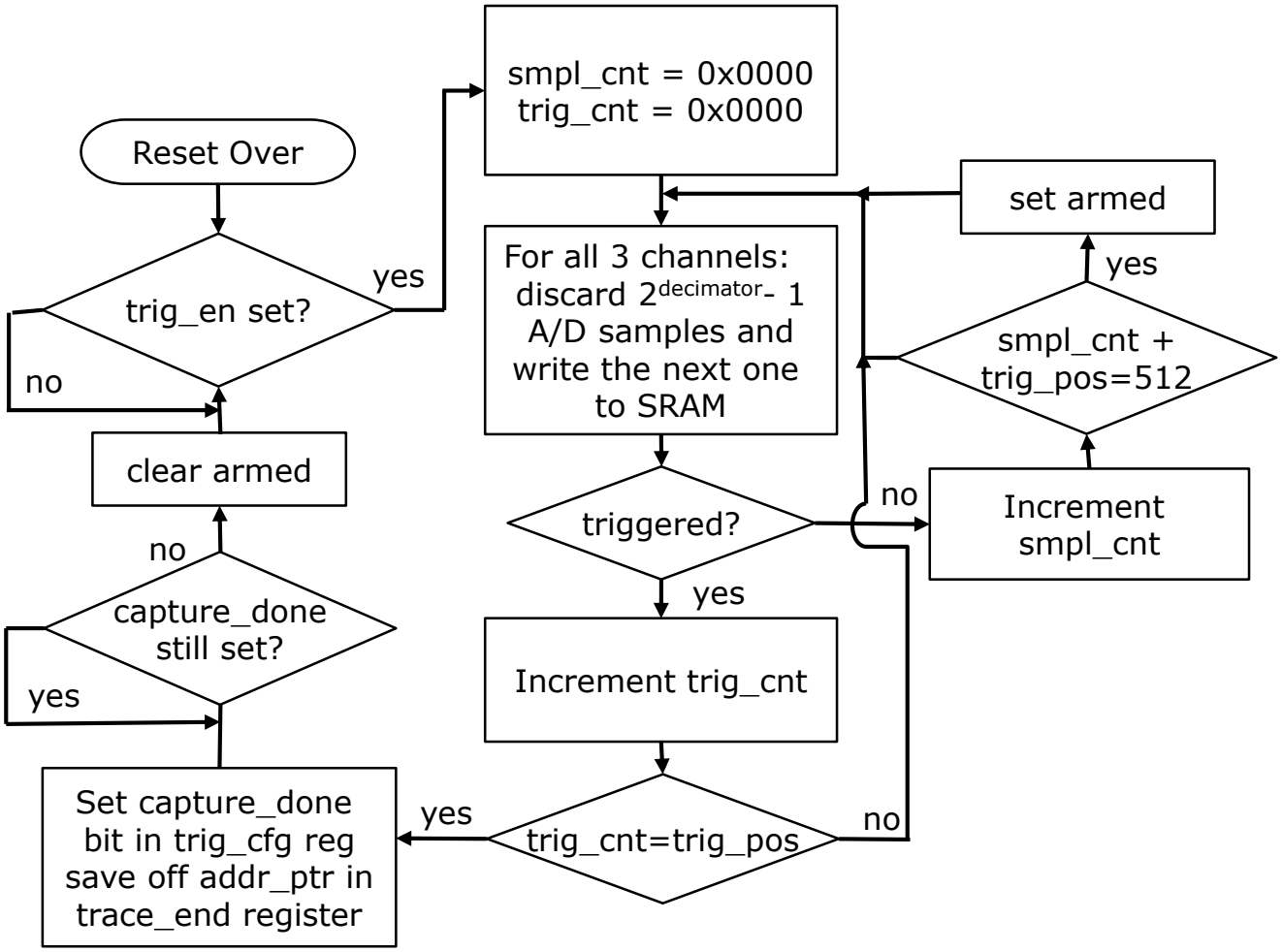
As long as the trigger mode is set in a run state, and the capture has not completed analog data from the 3 channels should be captured and stored in SRAM.

Upon a trigger event the current address pointer will be captured so the trigger point in the SRAM is known.

Capture continues until a configurable number of samples (trig_pos) after the trigger have been captured.

Once trig_pos number of samples after the trigger have been captured data acquisition ends and the "capture_done" bit is set.

"capture_done" is cleared by the command processing SM after the channel data is read.



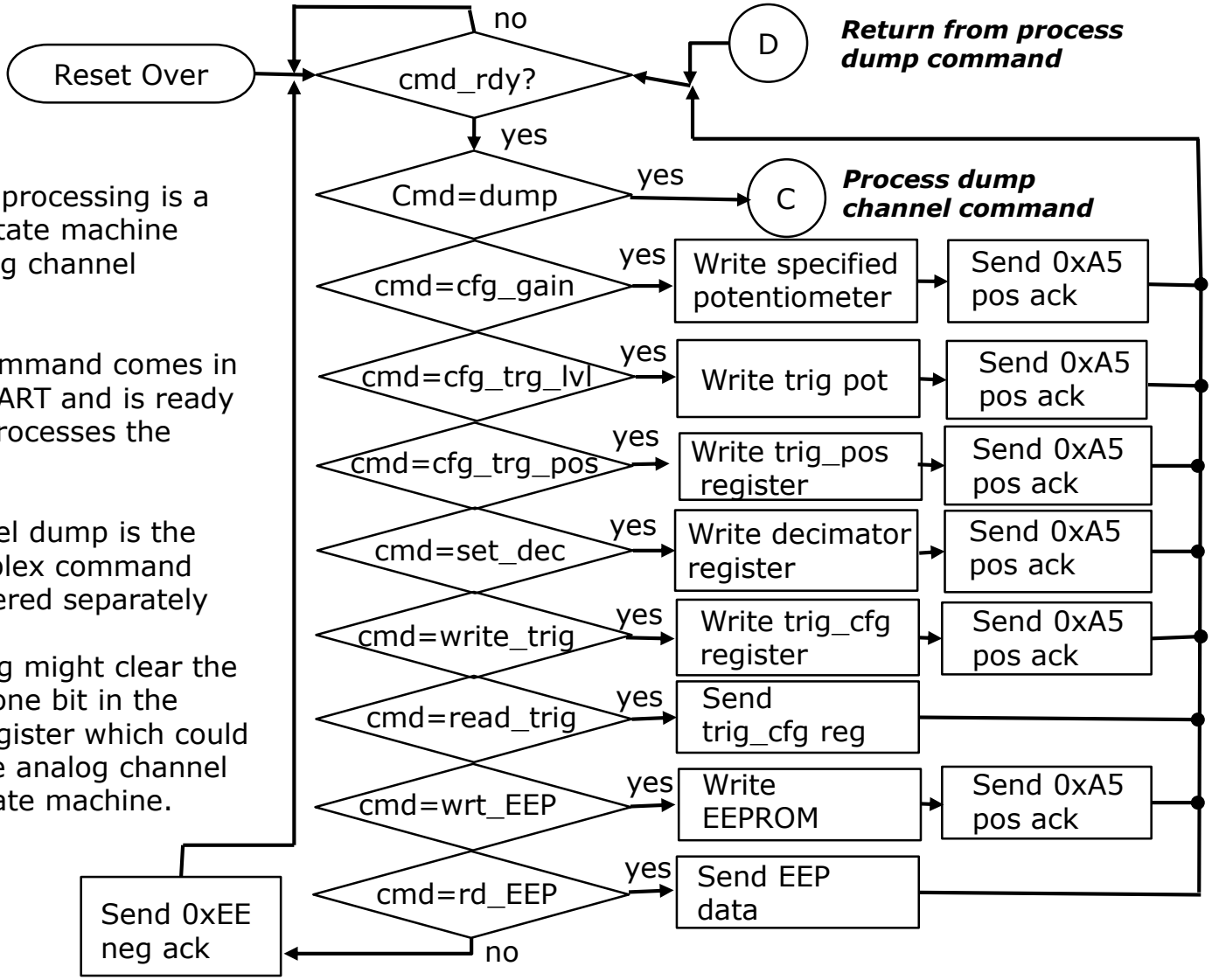
Digital Core Functionality (command processing)

Command processing is a separate state machine from analog channel capture.

When a command comes in over the UART and is ready it simply processes the command.

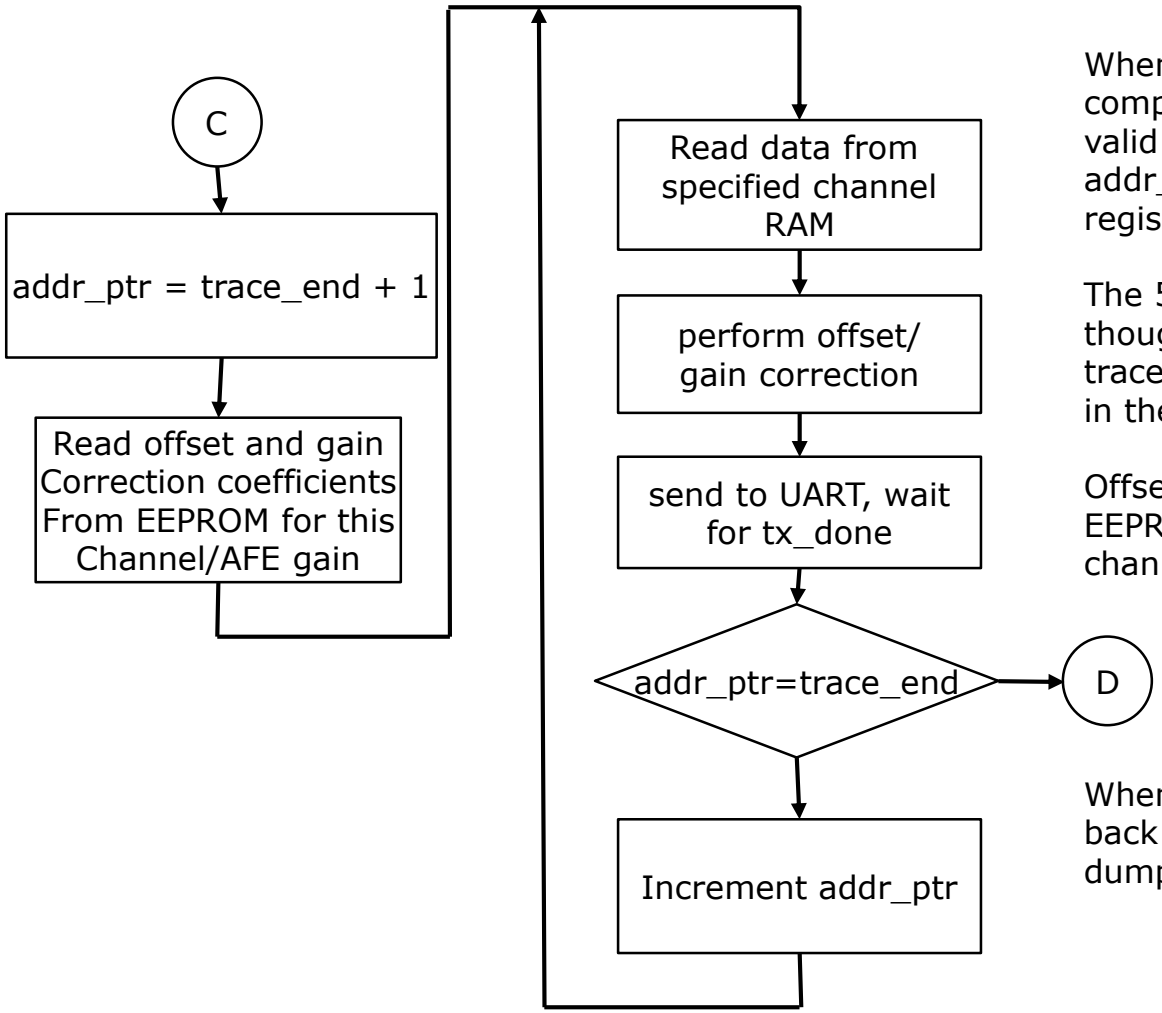
The channel dump is the most complex command and is covered separately

A write_trig might clear the capture_done bit in the trig_cfg register which could kick off the analog channel capture state machine.



Digital Core Functionality

- Dumping Channel Data



When the capture of the channel data completed addr_ptr was pointing to the last valid record of the trace. This value of addr_ptr was then stored off in another register called trace_end.

The 512 entries of the channel RAMs can be thought of as a circular queue. So trace_end+1 would point to the first record in the trace (earlier in time).

Offset and gain coefficients are stored in the EEPROM, and are used to correct the channel data as it is dumped to the UART.

When the addr_ptr wraps around and comes back to equal trace_end we are done dumping the data to the host.

Command Set (commands received over UART interface)

Command Encoding: All commands are 24-bits. The upper byte [23:16] encodes the opcode.

Bits[23:16]	Bits[15:8]	Bits[7:0]	Description:
8'h01	8'b000000cc	8'hxx	Dump channel command. Channel to dump to UART is specified in the lower 2-bits of the 2nd byte. **cc**=00 implies channel 1, **cc**=10 implies channel 3. and **cc**=11 is reserved
8'h02	8'b000gggcc	8'hxx	Configure analog gain of channel (this would correspond to volts/div on an opamp). Channel to set gain on is specified in lower 2-bits of the 2nd byte (cc). Analog gain value is specified by the 3-bit **ggg** field of the 2nd byte. See section **AFE Gain Settings** below for how this will translate to SPI commands. 3-bit registers storing the current gain for each will be used for accessing the proper calibration coefficients from EEPROM.
8'h03	8'hxx	8'hLL	Set trigger level. This command is used to set the trigger level. The value in the 3rd byte (8'hLL) determines the trigger level. Only values between 46 and 201 are valid.
8'h04	8'h0U	8'hLL	Write the trigger position register. Determines how many samples to capture after the trigger occurs. This is a 9-bit value
8'h05	8'hxx	8'h0L	Set decimator (essentially the sample rate). A 4-bit value is specified in bits[3:0] of the 3rd byte.

Command Set (continued)

Bits[23:16]	Bits[15:8]	Bits[7:0]	Description:
8'h06	8'b00dettcc	8'hxx	Write trig_cfg register. This command is used to clear the capture_done bit (bit[5] = d). This command is also used to configure the trigger parameters (edge, trigger type, channel)
8'h07	8'hxx	8'hxx	Read trig_cfg register. The trig_cfg register (described below) is sent out the UART.
8'h08	8'b00aaaaaa	8'hVV	Write location specified by 6-bit address of calibration EEPROM with data specified in the 3rd byte.
8'h09	8'b00aaaaaa	8'hxx	Read calibration EEPROM location specified by 6-bit address.

trig_cfg Register: bits[7:6] ➔ Always read zero

bit[5] ➔ capture_done bit. Set by capture state machine when capture done, cleared by host software after channel dump data has been acquired

bits[4] ➔ trigger edge. 1=> positive edge triggered, 2 => negative edge

bits[3:2] ➔ trigger type. 10 => auto roll, 01 => normal triggered, 00 => off

bits[1:0] ➔ trigger source. Only 00 and 01 are valid. 00 => channel 1 is source, 01 => channel 2 is source

AFE Gain Settings

- The channel amplifiers in the AFE have variable gain settings. The gain settings are varied through the use of digital potentiometers. These digital potentiometers are controlled through a SPI bus. There are 8 possible gain settings for each channel.

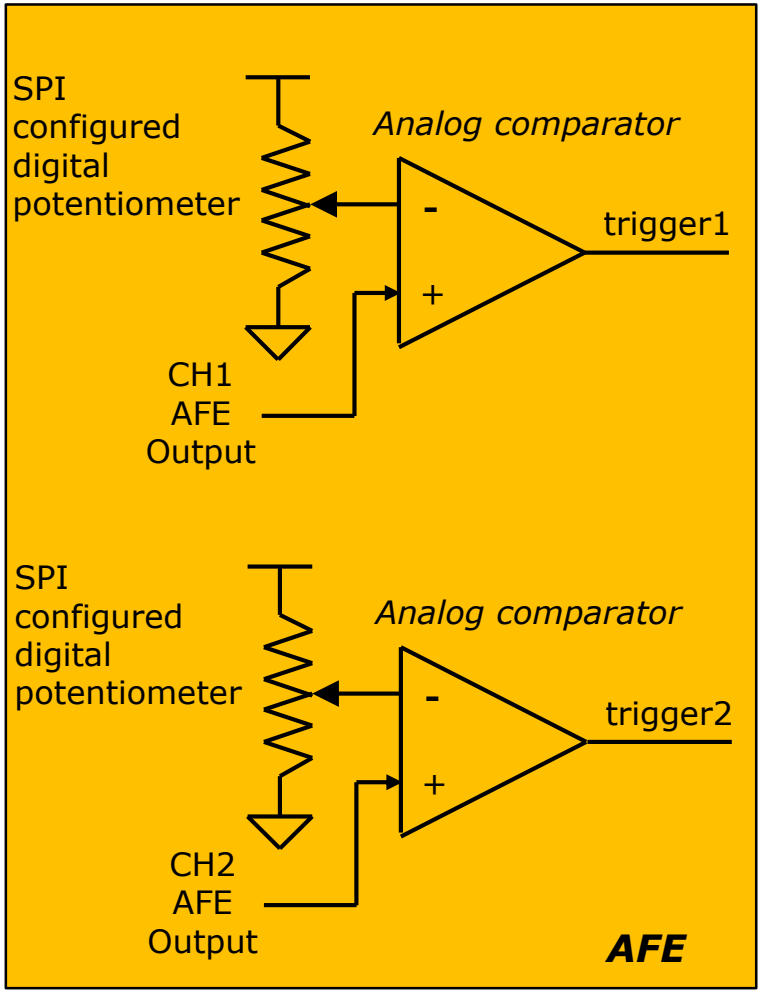
Gain (V/32codes)	ggg bits	16-bit SPI Command
50V	000	0x1302
30V	001	0x1305
20V	010	0x1309
10V	011	0x1314
5V	100	0x1328
3V	101	0x1346
2V	110	0x136B
1V	111	0x13DD

- The active low slave select (i.e. chX_ss_n) signals are used to specify which digital potentiometer is being set by the data on the SPI bus. ch1_ss_n would be held low while adjusting the gain for channel 1, and ch3_ss_n would be held low while adjusting the gain for channel 3

- The Gain in this table is specified as volts per 32 codes because 32 A2D codes will be considered 1 division on the display of the waveforms in the host application. So in effect the gain is given as volts/division similar to a normal scope.

- Looking back at the command encodings one can see that the second byte of the command from the host (8'b000gggcc) bits **cc** will determine which slave select signal is held low. Bits **ggg** wil specify one of the 8 analog gains.

Setting Trigger Level Over SPI



As can be seen in the diagram to the left, digital potentiometers are used to set the trigger level. These digital potentiometers are written via the SPI interface.

The SPI commands to the digital potentiometers are 16-bits wide. The upper byte of the command should be 0x13 to write to both potentiometers.

There is no need or reason to control the two potentiometers separately for trigger1 vs trigger2.

So the SPI command to send is:

{8'h13,cmd[7:0] }

Where cmd[7:0] is the lower 8-bits of the command received from the host, and carries the value to be written to the 8-bit digital potentiometer.

Digital Gain/Offset Correction

- As channel data is read and dumped to the UART offset and gain corrections are performed on the raw values in the digital domain.

$$Corrected = (Gain_{C\_G} * (Raw + Offset_{C\_G})) >> 7$$

  - The Gain and Offset values are 8-bit values obtained from the calibration EEPROM, and are specific for each channel, and analog gain setting (thus the: C_G subscript)
  - The Offset is a **signed** 8-bit value
  - The final result sent is the lower 8-bits of the 16-bit product after it has been shifted right 7-bits. A gain term of 0x80 would be unity.
  - The address of the gain term in EEPROM is: {**cc,ggg,og**} where **cc** is the 2-bit field specifying the channel, and **ggg** is the 3-bit field specifying the analog gain for that channel, and **og** is used to differentiate offset vs gain terms.

Address:	Description:
6'b000000	CH1 Offset for ggg=000
6'b010001	CH2 Gain for ggg=000
6'b010010	CH2 Offset for ggg=001
…	…
6'b101111	CH3 Gain for ggg=111

Contents of Calibration EEPROM

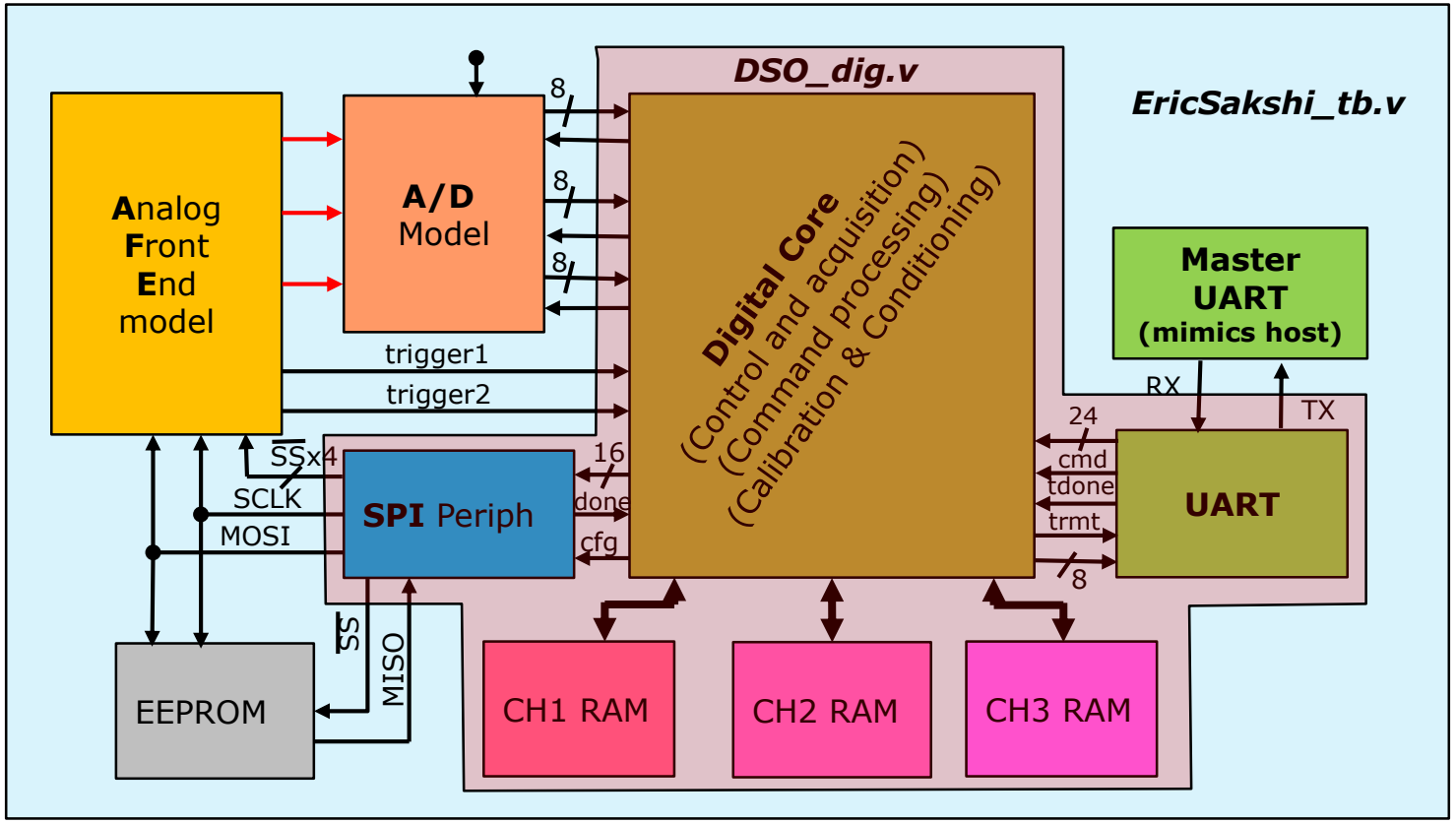
Calibration EEPROM

- The calibration EEPROM holds offset and gain coefficients used to correct raw channel readings as they are dumped out the UART port to the host PC.
- The calibration EEPROM is on the SPI bus and supports read and write operations. All packets to/from the EEPROM are 16-bits in width.
- Packet format for commands to EEPROM (what is sent on MOSI):

First Byte:		2nd Byte:
Bits[15:14] opcode	Bits[13:8] address	Bits[7:0] Data to write or don't care

- Opcodes are:
  00 => Read (data read comes on MISO in **next** SPI transaction)
  01 => Write (data to write is specified in low byte)
  1x => not implemented, reserved for future use
- NOTE: Reading a given EEPROM location requires 2 SPI transactions. The first SPI transaction sends the read command and address, but the data returned on MISO is garbage. The 2nd read transaction can specify another read command to a new address, and the data returned on MISO during that transaction is the data requested from the **previous** transaction.
- Therefore with 3 SPI transactions one can read the offset/gain coefficients needed to correct raw channel data as it is dumped out the UART.

Required Hierarchy & Interface



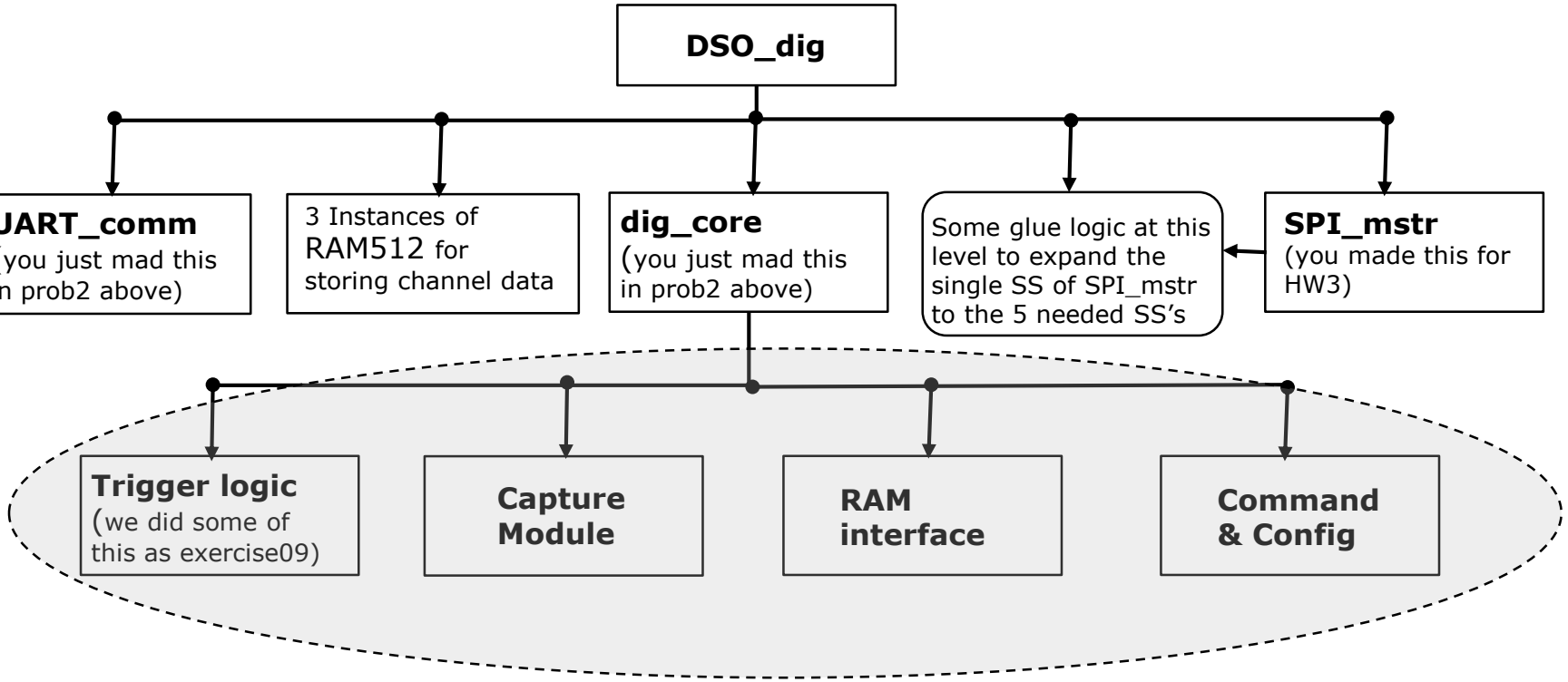
Your design will be placed in an "EricSakshi" testbench to validate its functionality. It must have a block called **DSO_dig.v** which is top level of what will be the synthesized DUT. The interface of DSO_dig.v **must match exactly** to our specified DSO_dig.v interface **Please download DSO_dig.v** (interface skeleton) from the class webpage.

The hierarchy/partitioning of your design below DSO_dig is up to your team.
The hierarchy of your testbench above DSO_dig.v is up to your team.

DSO_dig Interface

Signal Name:	Dir:	Description:
clk, rst_n	in	Clock input (40MHz), and active low async reset.
adc_clk	out	20MHz clock to the A/D converter
chX_data[7:0]	in	Data output from A/D converter for channel X, where X is one of: 1,2,3
trig1,trig2	in	The two trigger signals from CH1 & CH2. Used to initiate data capture to RAM
MOSI	out	Serial output of SPI bus
MISO	in	Serial input of SPI bus (only needed for reads of EEPROM)
SCLK	out	Clock of SPI bus (40MHz/16)
chX_ss_n	out	Active low slave select to the 3 digital pots used to configure channels 1,2,3
trig_ss_n	out	Active low slave select to digital pot used to configure trigger level
EEP_ss_n	out	Active low slave select to the EEPROM chip
TX	out	UART data out to UART2USB chip to HOST
RX	in	UART data in from UART2USB chip from HOST

Figure out your partitioning of ***dig_core***



Partitioning below ***dig_core*** is up to you guys. I partitioned into these 4 units. Does that portioning make sense? Trigger logic could have been part of the capture module. RAM interface could have been part of Command & Config. So could have had only 2 sub – units under ***dig_core***. You have to meet as a team, and discuss the partitioning of ***dig_core***.

Digital Core Interface (dig_core.v)

Signal Name:	Dir:	Description:
clk, rst_n	in	Clock input (40MHz), and active low async reset.
adc_clk	out	20MHz clock to the A/D converter. Can be left always running
trig1,trig2	in	The two trigger signals from CH1 & CH2. Used to initiate data capture to RAM
SPI_data[15:0]	out	Output command/select/data to digital potentiometers and EEPROM. Input to SPI peripheral
wrt_SPI	out	Control signal to initiate SPI transaction write/read
ss[2:0]	out	Determines which SPI slave is selected 000 => trig, 001-011 => chX_ss, 100 => EEP
SPI_done	in	From SPI unit. Indicates SPI transaction is completed
EEP_data[7:0]	In	Lower 8-bits of rd_data from SPI interface will carry bytes read from EEPROM.
rclk	out	20MHz clock to RAM block. Opposite phasing as ***adc_clk,*** can be left running.
en	out	Enable to RAM blocks. Hold low if not performing a RAM operation
we	out	Write enable to RAM blocks. Same ***we*** signal can be routed to all 3 RAM blocks since capture all 3 channel simultaneously.
addr[8:0]	out	RAM block Address to read/write to. Same address can be routed to all 3 RAM blocks
chX_rdata[7:0]	in	Read data from RAM block for channel X, where X is one of: 1,2,3

Continued next page

Digital Core Interface (continued)

Signal Name:	Dir:	Description:
cmd[23:0]	in	Command from host PC (in via UART, via USB)
cmd_rdy	in	From UART interface, indicates the 24-bit command is ready
clr_cmd_rdy	out	To the UART_comm module, asserted for 1 clock cycle to knock down ***cmd_rdy***
resp_data[7:0]	out	Byte to be sent to host PC via UART via USB.
send_resp	out	Asserted for one clock to UART interface to initiate transmission of ***tx_data[7:0]***
resp_sent	in	Asserted to indicate that UART interface is done transmitting ***tx_data[7:0]***

Provided Modules & Files:

- Interface & Model files **available on webside under: "Project as 551_DSO.zip"**

File Name:	Description:
DSO_dig_tb.v	**Optional** testbench template file.
DSO_dig.v	**Requried** interface skeleton verilog file. **Copy this** and flush it out with your design
dig_core.v	Optional digital core interface skeleton. Might be handy as starting point
AFE_A2Ds.v	Models the Analog Front End and the A2D converters in one file.
dig_pot.v	Model of digital potentiometer which is part of AFE model
SPI_EEP.v	Model of the EEPROM used to hold calibration offset/gain terms.
SPI_slv.v	SPI slave module that is part of both dig_pot.v and SPI_EEP.v
RAM512.v	Model of the RAMs used for storing raw channel data. 512 entries
ch1_analog.hex ch2_analog.hex ch3_analog.hex	Data file that represent the "analog data" that would be applied to the scope probes. You are free to also create your own for testing purposes. These are read by the $readmemh statement in AFE_A2Ds.v
CAL_EEP.hex	Represent initial gain and offset values stored in EEPROM. All unity gains and zero offsets. You should probably modify for better testing.