FERMILAB-SLIDES-25-0189-PPD-V

PSEC5 Physical Verification

H. D. Rico-Aniles, A. Fahey, P. Rubinov, A. Datta, A. Arzac, J. Park, R. Yeung, J. Pastika, H. Frisch, T. England, X. Wang, E. Angelico

Introduction

- Particle detection systems have achieved remarkable spatial resolution of 100μm or less

- As accelerator luminosity increases, such as at HL-LHC, spatial resolution alone is insufficient to distinguish between particles in dense environments.

- The addition of precise, 1ps level timing information adds a crucial fourth dimension to particle tracking, helping to resolve ambiguities in high-occupancy events and particle id using time-of-flight

- Among the most promising technologies in this frontier are Large Area Picosecond Photo-Detectors (LAPPDs).

- Realizing the full potential of these next-generation detectors will require parallel advancement in readout electronics, particularly the development of faster ASICs capable of measuring increasingly precise timing signals.

- PSEC5 is an ASIC that has been designed as the core of the readout system to achieve 1ps sampling time resolution on an LAPPD sensor.

- The efforts of this project were to physically validate the design of PSEC5 ASIC on a chip-on-board PCB.

Architecture of PSEC5 sampling

- 8 Channel multi-hit design.
- 4 Fast switched-capacitor arrays (SCA).
- 1 Slow SCA.
- Zero crossing discriminator.
- VCO as clock source.
- SPI communication.

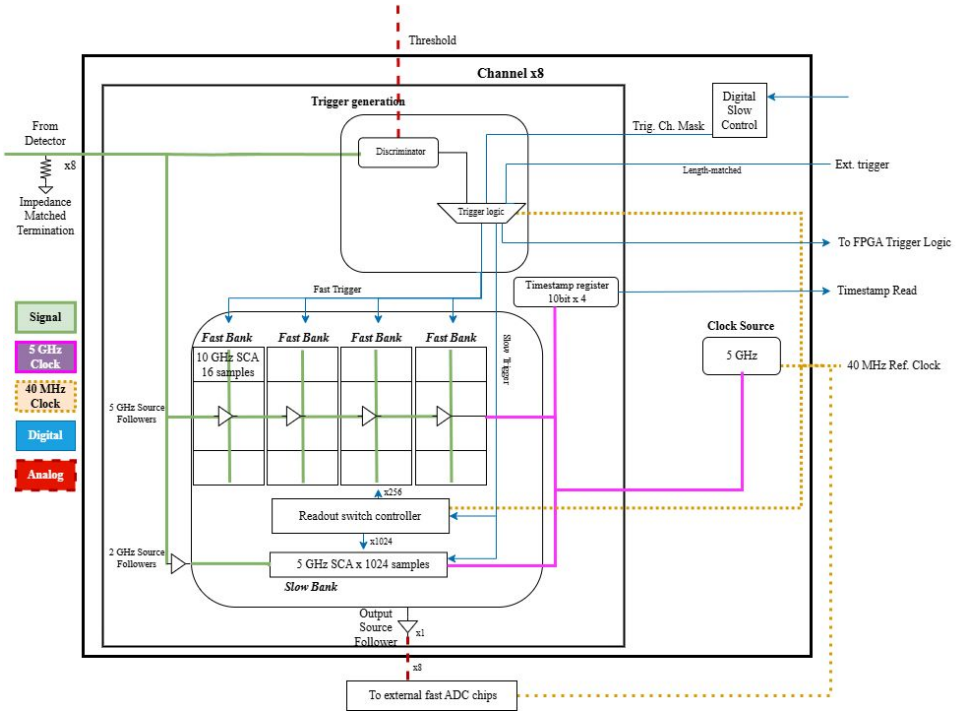

Figure 1. PSEC5 block diagram.

Testing PCBs

Three PCBs were designed to test this chip.

1. DUT board
2. Control board
3. Chip on board (COB)

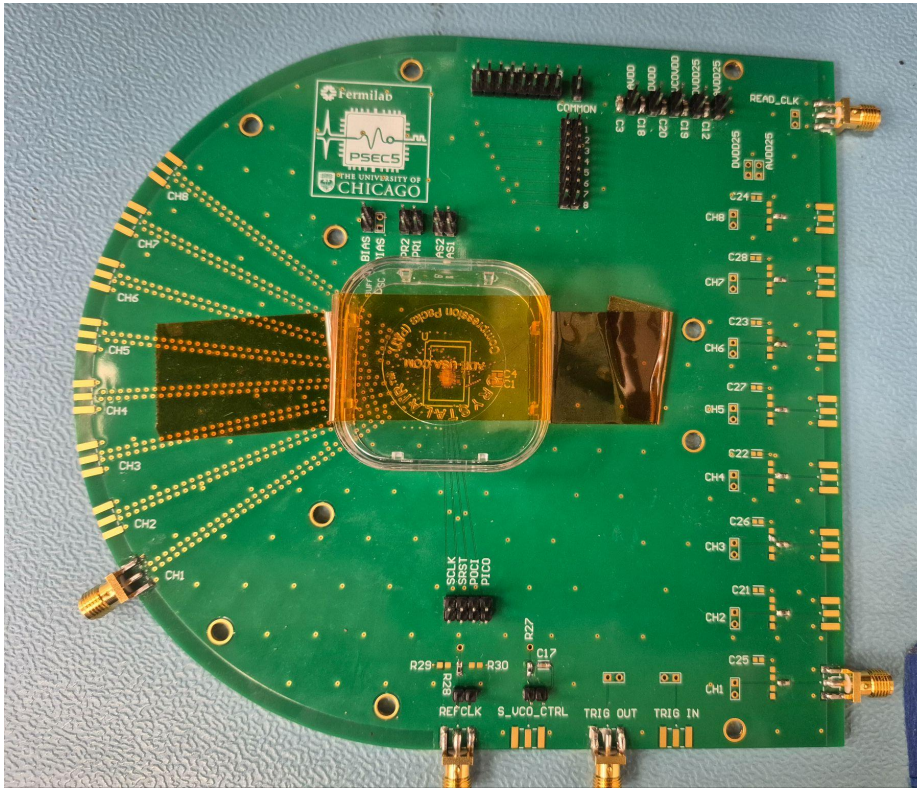

Figure 2. COB PCB.

Packaged Chip Standalone Board

The COB board was modified to test packaged PSEC5 chips. See figure 3.

The package is a 72 lead QFN package

The leads of the QFN package on the footprint were extended ~0.5mm to facilitate soldering.

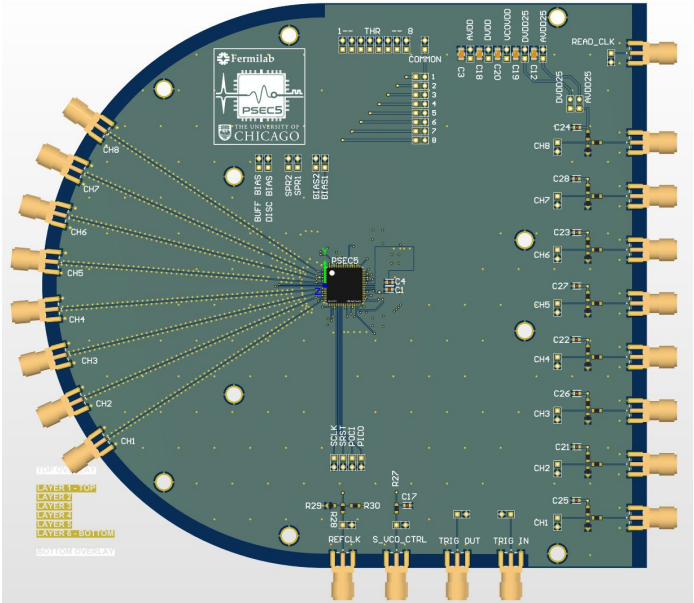

Figure 3. Packaged PSEC5 standalone testing board.

Testing Methodology

1. Wire bonding visual inspection
2. PCB visual inspection
3. Power up test
4. SPI
    a. COPI, CIPO, CS
5. Readout buffer
6. Instruction register
7. Read counter registers
8. Clock divider
9. VCO
    a. VCO capacitor band register
    b. Frequency tuning
    c. Supply variations
    d. Jitter
10. Discriminator
11. Readout SCA

Wire Bonding Visual Inspection

- Pin N3 on the PCB is connected to DVDD and should be connected to DVDD25. See Figure 5.
- Pin E14 on the bonding diagram has a double connection to READBIAS1 and to GND. The pin is properly connected to BIAS1 on the PCB.
- Pin E17 on the bonding diagram should be wirebonded to VCODVDD instead of DVDD.
- Pin N13 on the bonding diagram should be connected to AVDD instead of DVDD.

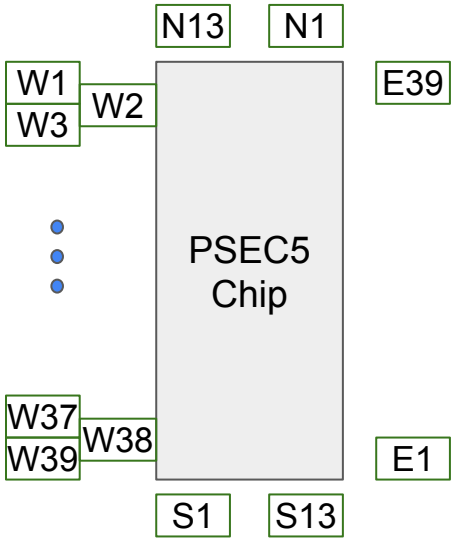

Figure 4. Pad identification format.

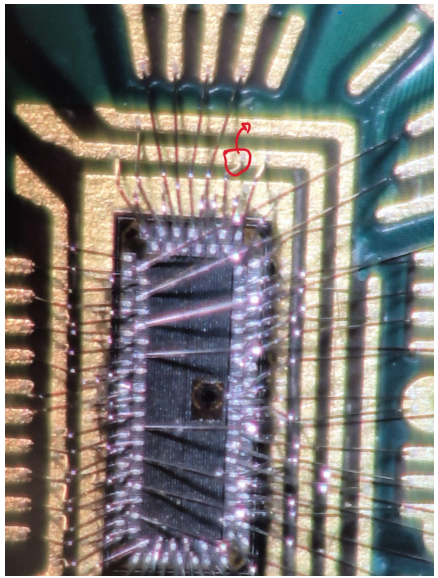

Figure 5. Required wired bond modification.

Bonding Diagram Updated Naming

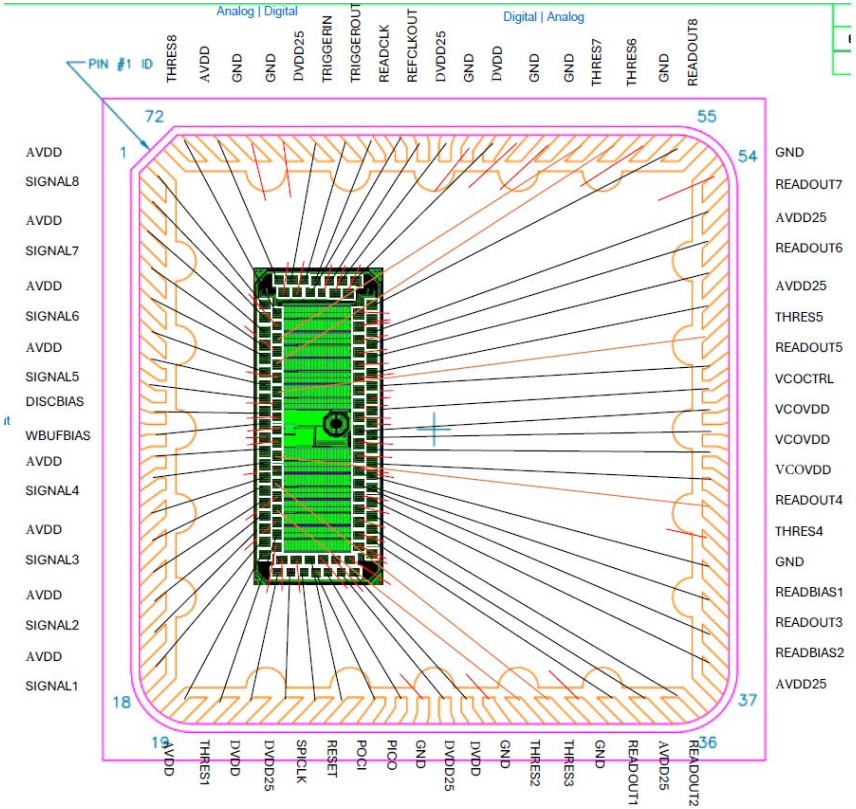

Figure 6. Bonding diagram with updated supplies naming.

PCB Visual Inspection

On the PCB design there are 4 floating ground connection on the south side wire bonds of PSEC5.

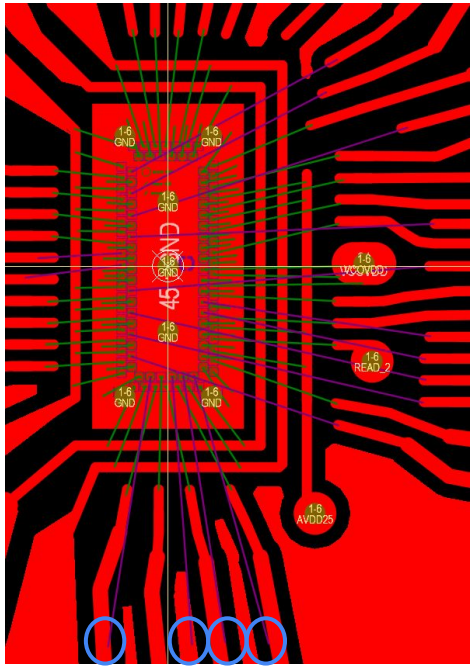

Figure 7. Floating wire bond grounds on COB.

Power Test

- The board has 5 different supply voltages: AVDD, DVDD, VCOVDD, AVDD25, and DVDD25.

- There were three short circuit on the board.
  - DVDD25 connected to DVDD wirebond N3.
  - VCOVDD connected to DVDD wirebond E17.
  - AVDD connected to DVDD wirebond N13.

- The quiescent power consumption is ~21.4mW

Table 1. Quiescent power.

Supply	Nominal Voltage (V)	Current (mA)	Power (mW)
AVDD	1.2	0.6	0.72
DVDD	1.2	2.0	2.4
VCOVDD	1.2	~6.09	7.308
DVDD25	2.5	0.05	0.125
AVDD25	2.5	4.34	10.85

Signal Buffer Power Test

- It is a source follower buffer used to interface the input signal with the switched-capacitor arrays.
- There are 4 fast buffers (~5GHz) and 1 slow buffer (~2GHz) per channel.
- They are powered from the AVDD supply and biased simultaneously through the buffer bias pin.
- The bias voltage is applied directly to the tail biasing transistor.

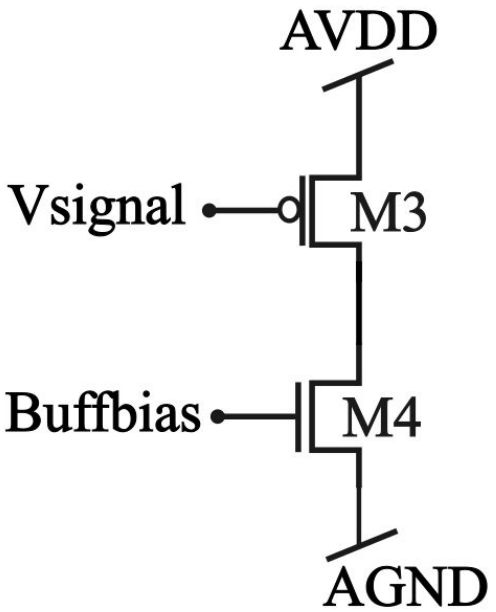

Figure 8. Signal buffer schematic.

Signal Buffer Power Test

When biased, the total quiescent current drawn from AVDD increases up to ~85mA.

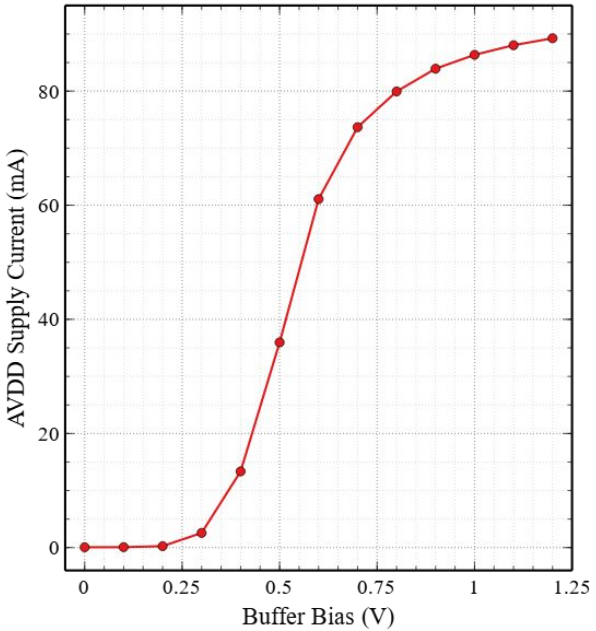

Figure 9. Quiescent current drawn by input buffer when biased voltage is applied.

SPI Testing

- The SPI controller was an Arduino Pro Portenta H7 board.
- The microcontroller firmware gives the user 3 options that can be selected by the numbers 0-2.
  - 0 - external reset.
  - 1 - send a hexadecimal command
  - 2 - set up a cycle between two hex commands.
- Commands are 2 bytes sent in hexadecimal. Example: 3F01

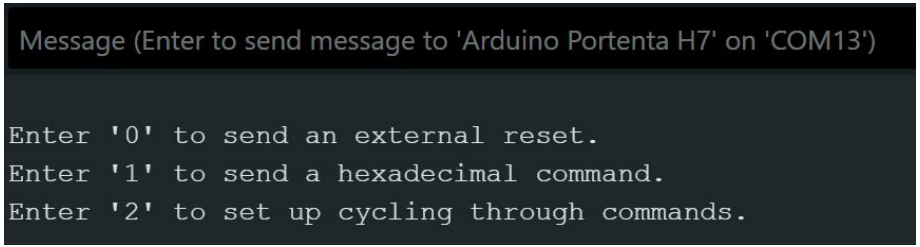

Figure 10. Arduino serial monitor interface to control SPI.

SPI Testing

- COPI
  - The SPI on the PSEC5 is capable of receiving data in two bytes format from an external controller.
- CIPO
  - the PSEC5 responds with the value stored on the register of the address sent.
  - 1 bit, out of 16, is lost and data is shifted right because is not a standard SPI and requires 17 clock cycles.
- CS
  - CS of common SPI implementation is used as the external reset for the chip.

Register Map

Register Address	R/W	Content	Description
0		Reserved	Will read out nothing
1	R/W	Trigger Channel Mask	0: Disabled, 1: Enabled
2	R/W	Instruction (2 bits)	1: Reset, 2: Readout, 3: Start
3	R/W	Mode (2 bits)	00: Use 1 Fast SCA bank to capture an edge, 01: Use 2, 10: Not Used, 11: Use 4.
4-10	R	Counter 0 Values	structure: {000, trigger_cnt, CE, CD, CC, CB, CA} (MSB to LSB)
11-17	R	Counter 1 Values	
18-24	R	Counter 2 Values	
25-31	R	Counter 3 Values	
32-38	R	Counter 4 Values	
39-45	R	Counter 5 Values	
46-52	R	Counter 6 Values	
53-59	R	Counter 7 Values	
60	R	PLL Locked (1 bit)	Only necessary with onboard PLL
61	R/W	Discriminator Polarity (8 bits)	0: Falling Edge. 1: Rising Edge
62	R/W	VCO Digital Band (6 bits)	
63	R/W	PLL Division Ratio(5 bits)	512, 256(40MHz if 10GHz VCO), 128, 64, 32. only leading bit matters
64	R/W	Slow mode (1 bit)	Uses 2.5GHz clock instead of 5GHz clock
65	R/W	Trigger Delay (6 bits)	Discriminator output to trigger is delayed by this amount of clock cycles. (Max 31)

Writing to the PSEC5 Registers

The SPI communication with PSEC5 is achieved by sending two bytes of information.

- First byte is the address
- Second byte is the data to be written to the register.
- Example: to select the division by 512 the value 1 should be written to register 63. This is achieved by sending the SPI hex data 0x3F01.

The communication follows the timing diagram on the figure below. Serial out should return the data previously stored in the register.

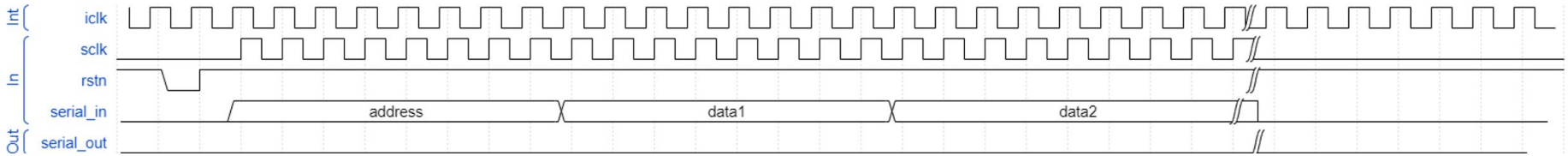

Figure 11. Timing diagram to write to a register.

Reading from the PSEC5 Registers

- Reading from a register follows the same procedure as writing to a register with the difference that the 2nd byte (data to be stored during writing) is ignored.
- PSEC5 is a non-standard 17bits SPI communication.
- A standard 16bits SPI communication was implemented leading to the lost of the LSB.

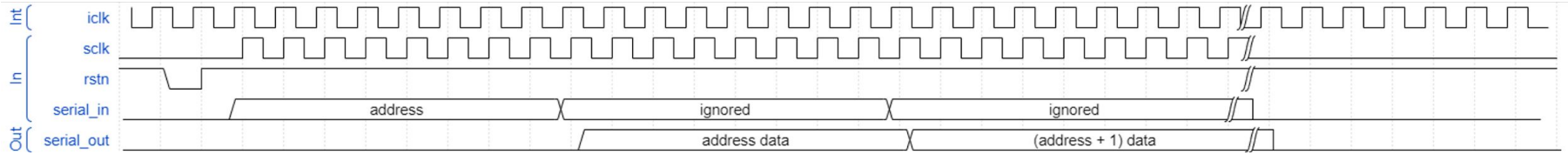

Figure 12. Timing diagram to read from a register.

Readout Buffer

- The readout buffer is used to read the voltages stored in the capacitors of the SCA.
- Per channel, there are 4 buffers to read the fast banks and 1 buffer to read the slow banks.
- Two biasing voltages are used for these buffers, Readbias1 and Readbias 2, applied directly to the gates of M2 and M1, respectively.
- Transistor M1 is common to all readout buffers in the channel, while each buffers has its own transistor M2.

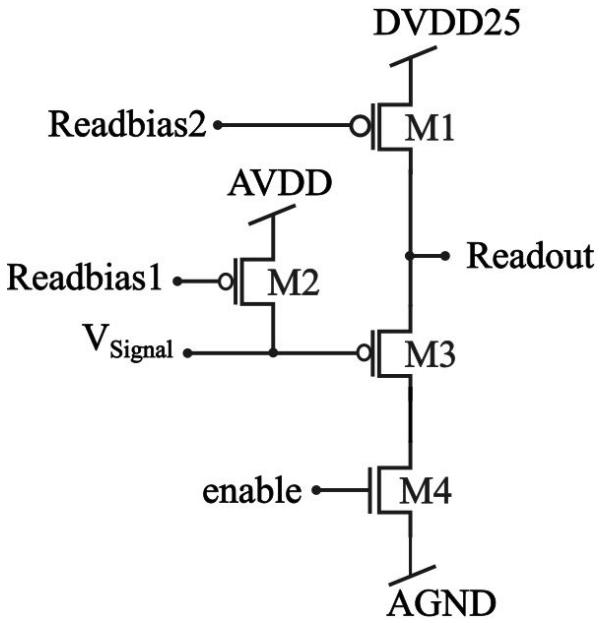

Figure 13. Readout buffer schematic.

Readout Buffer Enable Logic

The enable is generated by an additional logic.

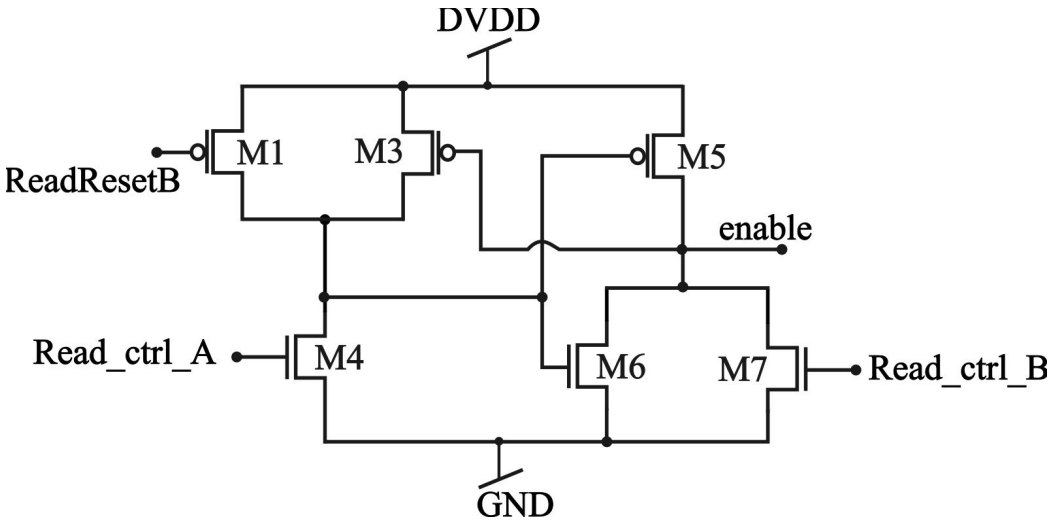

Figure 14. Readout buffer enable logic schematic.

Readout Buffer Bias Test

Procedure

1. Give an external reset
2. Set the clock divider
3. Send a mask value of FF
4. Turn ON input signal (Sinusoidal wave 1kHz, 1.2Vpp, offset 0.6V)
5. Bias buffer to 0.6V.
6. send the instruction START
7. Send the instruction READOUT
8. Turn ON the read clock (square wave 1Hz, 2.5Vpp, offset 1.25V)
9. Sweep both read bias (bias1 and bias2 on the board) values from 0.9 down to 0.5 V
10. Observe channel 1 output.
11. Observe for current changes.

Observations

- AVDD current raised to ~68.3mA on step 5.
- DVDD goes from 6 to 43.5mA on step 6.
- DVDD goes to ~1.6mA and jumps between 1.6 and 1.78mA with readclock on step 7.
- Also on step 7, the reference clock goes high, and the SPI become unresponsive.
- The output always reads high (2.5V).

Instruction Register Test

There are 3 different instructions that can be written to register 2

1. Start - command 0x0203
2. Readout - command 0x0202
3. Reset - command 0x0201

Observations

- START instruction increases the current by ~37.5mA of DVDD supply.
- READOUT makes the reference clock go high and makes SPI unresponsive
- Unknown effect of this instructions.

Read Counter Registers

- Reading counter registers when a READOUT instruction has not been given, they always return 0xFF.

- If a READOUT instruction is given, the reference clock output goes high and the SPI is unresponsive; register cannot be read.

- Sending a RESET instruction or and external RESET does not change the value 0xFF from the counter registers.

Clock Divider Test

- The clock divider exhibits 5 clock divisions (32, 64, 128, 256, and 512).
- These are achieved by writing register 63.
- Only 5 out of the 8 bits sent are used.

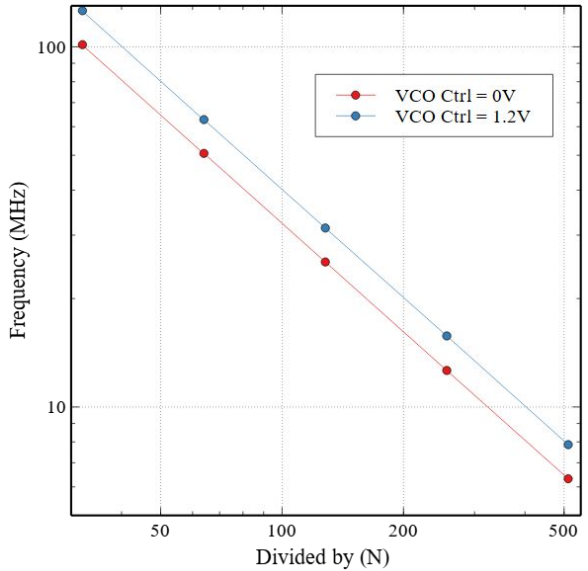

Figure 15. Log-log graph of frequency variation with clock divider control. VCO band = 0x00.

Clock Division Structure

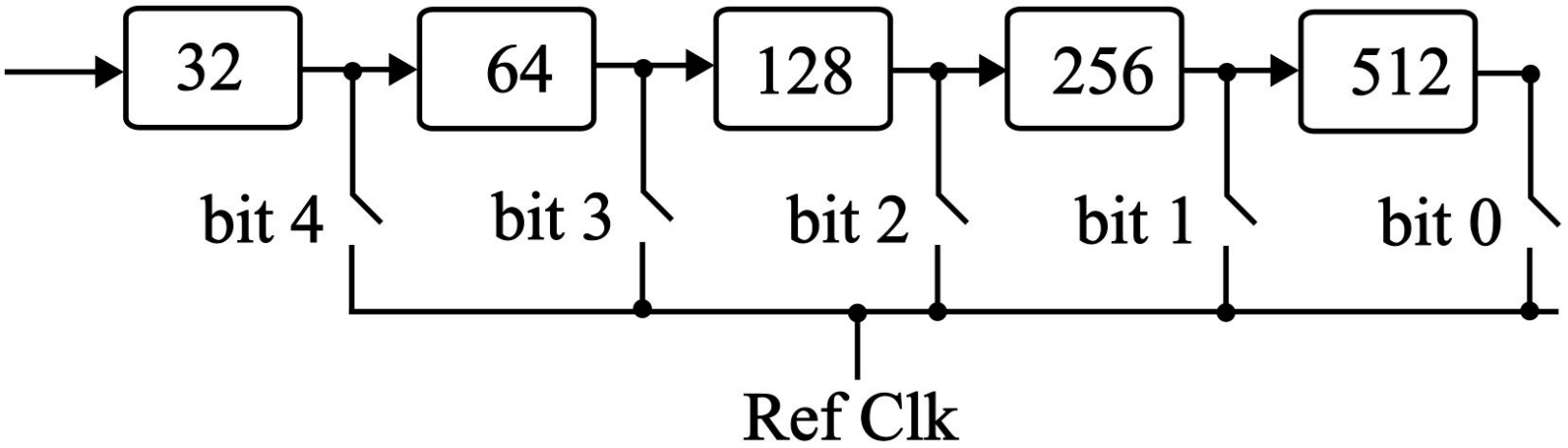

Figure 16. Structure to divide the clock.

The VCO testing

- The VCO oscillates within a frequency range of 3.2 - 4.04 GHz

- Fine frequency tuning is achieved through VCOctrl voltage (0-1.2V). See Figure 17.

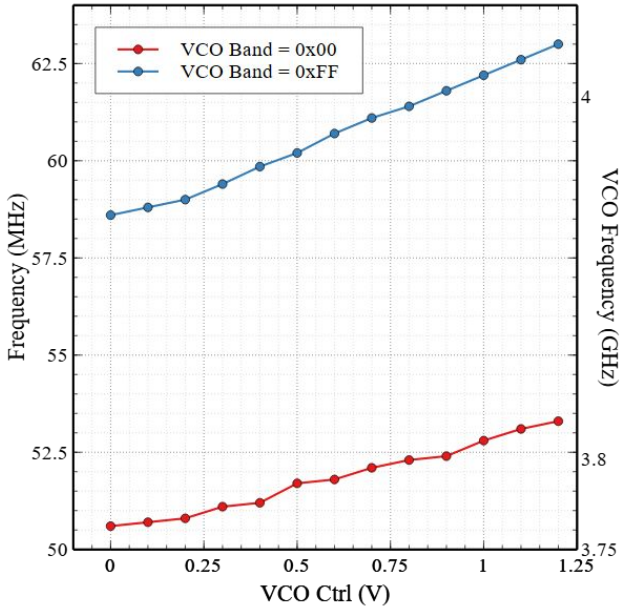

Figure 17. Frequency variation with clock divided by 64.

The VCO Frequency Variation with VCO Band

Coarse tuning of the frequency is achieved by writing to register 62 (VCO capacitor bank).

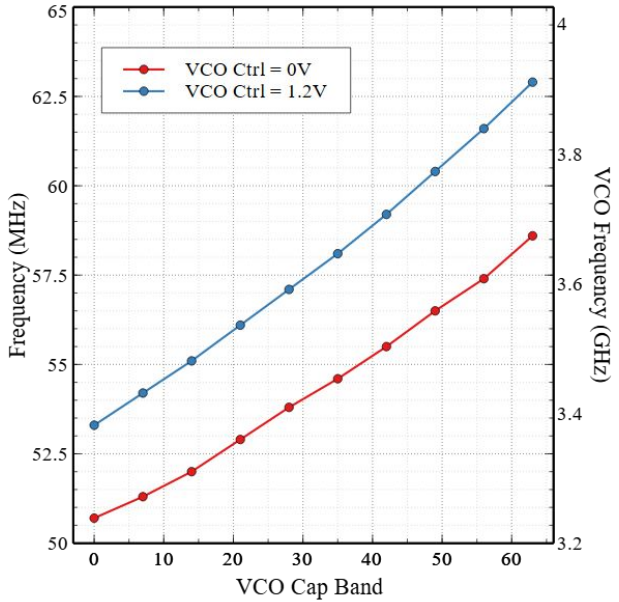

Figure 18. VCO Frequency vs VCO band with clock divided by 64.

VCO Supply Variations and Current drawn

- The VCO is power by an independent supply source VCOVDD.
- Max current drawn is 6.1mA at the lowest frequency.

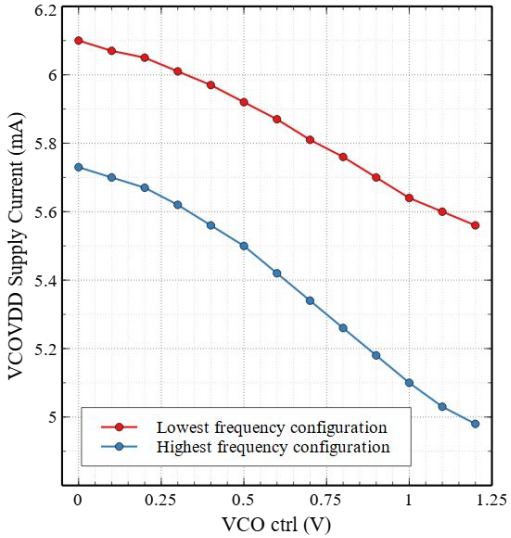

Figure 19. VCOVDD current variation with VCO ctrl. Lowest freq conf. VCO band =0x00 and clk div =512. Highest freq conf. VCO band =0xFF and clk div =32.

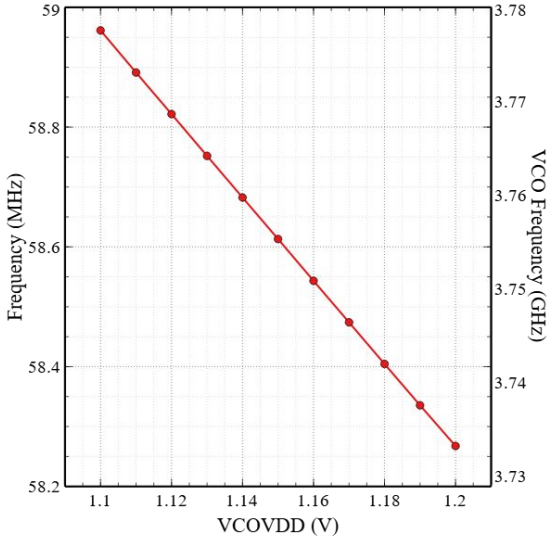

Figure 20. VCO frequency vs power supply variations.

VCO Jitter

- The jitter was approximated by measuring the variation of the rising edge of one clock cycle compared to an n^{th}-clock cycle after it.

- The period of the clock was T=~17.24ns and the measurements were taken for the 1, 2, 4, 8,...1024 that represent the time T, 2T, 4T,..., 1024T with a fixed value VCOCtrl=0V.

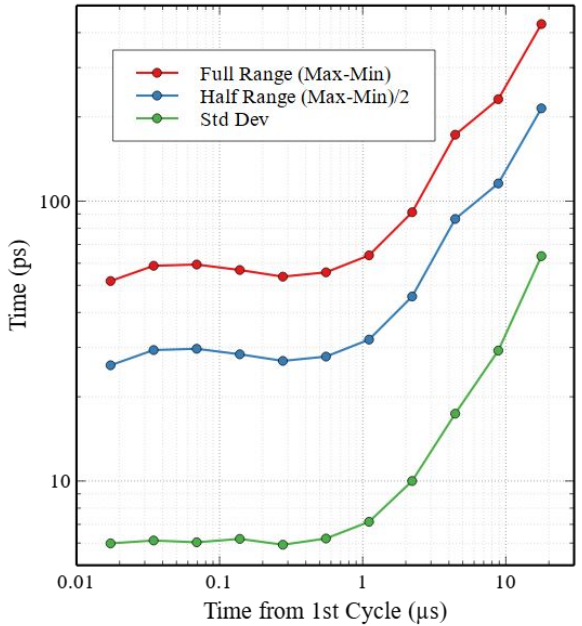

Figure 21. Variation for rising edge of two different clock cycles with VCOCtrl=0V

The Discriminator

- Receives the input signal and threshold directly from the detector and a voltage source, respectively.

- It is power by the DVDD power supply.

- The biasing voltage is directly applied to the gate of a tail biasing transistor.

Discriminator Test

The test performed on the discriminator followed the steps on the right.

DISC BIAS seems to not have any effect on the current drawn from supplies.

No trigger was read at the trigger out pin.

1. Power ON the board.
2. Bias the discriminator and buffer with 0.6 V
3. Turn on all power supplies
4. Apply 0.6V to threshold 1
5. Send an external reset to the board
6. Write to register 63 (clock divider) a value 0x01.
7. Write to register 62 (VCO band) a value, try it with 0x00 first.
8. Write to register 1 (Channel Mask) the value 0xFF.
9. Connect the trigger of the oscilloscope to the channel reading Trig out output.
10. Turn on the output of the function generator to square wave with 50% duty cycle, 1.2Vpp, offset of 0.6V, and frequency of 100kHz.
11. Send the instruction START to register 2.
12. Document the observation on oscilloscope, POCI, and currents from supplies.
13. Repeat for different values on register 62. Do not forget to do one test with the value FF in this register.
14. Document observations.

Proposed Readout from SCAs

Test conditions

- Set Disc. thresh to 700mV.
- Set Disc. Bias and input buffer Vbias to 490mV
- Set Readbias1 to 600mV
- Set Readbias2 to 1.4V
- Channel 1 Input signal 100kHz, 1.2Vpp, and 0.6V offset
- Read Clk is 1MHz 2.5Vpp 50% duty cycle Square wave with 1.25V offset.

Procedure

1. Send a reset
2. Set the clock division ratio to divide 128 (SPI data 0x3F04)
3. Set the VCO digital band to 00011100 (SPI data 0x3E1C).
4. Set slow mode to 0 (SPI data 0x4000)
5. Set trigger mask to 01111111 (SPI data 0x017F)
6. Set mode to 00000010 (SPI data 0x0302)
7. Set disc polarity to 11111111 (SPI data 0x3DFF)
8. Send the instruction START to register 2.
9. Send and external trigger.
10. Send the instruction READOUT to register2.
11. Read with the oscilloscope the output of Channel 1.

Things known good about the chip

- It powers up with a total ~21.4mW quiescent current; buffers are not biased.
- SPI PICO works
  - Data can be written to the R/W registers
  - VCO band, Clock Divider, Channel Mask, and instruction can be controlled
  - Data can be read from ALL registers

- VCO
  - oscillates ~3.2 - 4.04GHz frequency
  - Robust to other (not VCOVDD) supply voltages.
  - Can be controlled with the VCO capacitor bank and VCO control voltage.

- Buffer is assumed to be working since it draws current when applying a biasing voltage.

Things Understood to be bad about the chip

- Lower VCO frequency 3.2 - 4.04 GHz instead of ~10GHz.
- Discriminator bias does not draw current from any of the supplies when a biasing voltage is applied.
- The chip will receive the instruction START and followed by instruction READOUT only one time.
  - Readout makes the reference clock to be high.
  - SPI becomes unresponsive after a Readout instruction.

Discrepancies Between Simulation and Physical Test Chip

- VCO frequency in simulation is ~10GHz but ~4GHz in the physical chip; Simulation was perform on the VCO block independently.

Things not understood

- The discriminator does not seem to get biased.
- Trigger output does not toggle.
- Readout from the SCA does not output anything, yet.

Conclusions

The PSEC5 was tested using a chip-on-board (COB) PCB. The physical validation of the VCO and SPI communication was successfully validated. Operation of the input signal buffer was partially validated.

Read only register seem to not change their value of 0xFF.

Validation of the discriminator, switched-capacitor sampling banks, and readout buffer was not possible. Simulation of the system is required to understand the reason of the observed behaviour or to provide the appropriate steps on these blocks are operated.

Future Work

- To simulate the overall system to provide guidance for the physical testing and understand discrepancies or not working states of the physical chip.

- To design a one-channel ASIC with more pins available as test points to validate full functionality of building blocks of PSEC5.

- To organize the design and operation documentation of PSEC5.

Acknowledgements

- To the University of Chicago and Fermilab PSEC5 research group.

- This manuscript has been authored by FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics

- This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Visiting Faculty Program (VFP).

Appendix A - Arduino code

```
//Code for testing SPI operation
//2025.07.03
//Alexander Fahey


#include "Arduino.h"
#include "mbed.h"

using namespace mbed;

SPI spi(PC_3, PC_2, PI_1);  // MOSI, MISO, SCK, CS* (optional).
char hexString[5];
int value;
int i;
short CycleLowValue;
short CycleHighValue;
int CycleSpeedValue;
uint16_t tx_data;
uint16_t rx_data;

//The following code is meant to allow some simple tests of the SPI that can be run in succession without reuploading the code with minor changes.
//Basic Operation
//Give user 3 choices that user can select between:
//1) Send the reset signal and automatically return to the menu.
//2) Allow user to send one command in hex format and automatically return to the menu.
//3) Allow user to cycle through set of possible commands and set begin loop, end loop, and cycle speed
  //Also, show end command while cycling to allow returning to menu.
```

```
void setup() {
  Serial.begin(20000000);     // Initialization of serial communication.
  spi.frequency(10000000);   // Set up frequency.
  spi.format(16, 0);         // Message length (bits), SPI_MODE - check these in your SPI device's data sheet.
  MyReset();
}

void MyReset() {
  //Sends a 1ms high pulse on the Chip Select. This is connected to SRST to serve as an Active-High external reset.
  Serial.println("SPI Reset");
  digitalWrite(PI_0, HIGH);  // Drive the CS HIGH.
  delay(1);
  digitalWrite(PI_0, LOW);  // Drive the CS LOW.
  delay(1);
}

void MyCommand() {
  //Waits for 4 chars the user should send in due to early prompt and then puts them in an array.
  //This new array of 4 chars is translated into a hex value and written to the SPI.
    //Chars are assumed to be ones with Hexadecimal equivalents (e.g., 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
  while (true) {
    if (Serial.available() > 3) {
      break;
    }
    delay(500);
  }
  i = 0;
  while (i < 4) {
    hexString[i] = Serial.read();
    Serial.println(hexString[i]);
    i += 1;
  }
  value = strtol(hexString, NULL, 16);
  Serial.println(value, HEX);
  spi.write(value);
}
```

```
void MyCommandWithResponse() {
  //Waits for 4 chars the user should send in due to early prompt and then puts them in an array.
  //This new array of 4 chars is translated into a hex value and written to the SPI.
  //Chars are assumed to be ones with Hexadecimal equivalents (e.g., 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
  while (true) {
    if (Serial.available() > 3) {
      break;
    }
    delay(1);
  }
  i = 0;
  while (i < 4) {
    hexString[i] = Serial.read();
    Serial.println(hexString[i]);
    i += 1;
  }
  tx_data = strtol(hexString, NULL, 16); //translates an array of ASCIIs to hex value
  Serial.print("Data to be sent: ");
  Serial.println(tx_data, HEX);
  rx_data = spi.write(tx_data);
  Serial.print("Data recieved: ");
  Serial.println(rx_data, HEX);        //sends a command to spi and reads the response to the command before that
}
```

```
void SetCycleLowValue() {
  //Waits for 4 chars the user should send in due to early prompt and then puts them in an array.
  //This new array of 4 chars is translated into a hex value and saved for the cycling command.
    //Chars are assumed to be ones with Hexadecimal equivalents (e.g., 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
  while (true) {
    if (Serial.available() > 3) {
      break;
    }
    delay(500);
  }
  i = 0;
  while (i < 4) {
    hexString[i] = Serial.read();
    Serial.println(hexString[i]);
    i += 1;
  }
  CycleLowValue = strtol(hexString, NULL, 16);
  Serial.readString();
}
```

```
void SetCycleHighValue() {
  //Waits for 4 chars the user should send in due to early prompt and then puts them in an array.
  //This new array of 4 chars is translated into a hex value and saved for the cycling command.
    //Chars are assumed to be ones with Hexadecimal equivalents (e.g., 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
  while (true) {
    if (Serial.available() > 4) {
      break;
    }
    delay(500);
  }
  i = 0;
  while (i < 4) {
    hexString[i] = Serial.read();
    Serial.println(hexString[i]);
    i += 1;
  }
  CycleHighValue = strtol(hexString, NULL, 16);
  Serial.readString();
}
```

```
void SetCycleSpeedValue() {
  //Waits for chars the user should send in due to early prompt and then converts them into a decimal value and saves it to a variable.
    //Chars are assumed to be ones with decimal equivalents (e.g., 0,1,2,3,4,5,6,7,8,9)
    //-----Also, apparent speed is clearly different from this value. Not sure what I missed, but I will likely find the issue and fix it later.
  while (true) {
    if (Serial.available() > 1) {
      break;
    }
    delay(500);
  }
  String cmdstr = Serial.readString();
  Serial.println(cmdstr);
  CycleSpeedValue = cmdstr.toInt();
  Serial.println(CycleSpeedValue);
}
```

```
void MyCyclingCommand() {
  //Uses accumulated data to write every value between the low and high value to the SPI with the speed modulating
  //how quickly it does s0. This process will loop back to the low value once it reaches the high value.
  value = CycleLowValue;
  while (true) {

    //Serial.println("Enter '0' to exit Loop.");       //if you uncomment this block you will be able to exit the loop without reseting but it will be much slower
    //String cmd = Serial.readStringUntil('\n');
    //if (String(cmd) == "0") {
    //  break;
    //}

    if (value == CycleHighValue) {
      value = CycleLowValue;
    }
    Serial.println(value, HEX);
    spi.write(value);
    value += 1;
    delay(CycleSpeedValue);
    //receivedData = SPI.transfer(dataToSend);
  }
}
```

```
void loop() {
  Serial.println();
  Serial.println("Enter '0' to send an external reset.");
  Serial.println("Enter '1' to send a hexadecimal command.");
  Serial.println("Enter '2' to send a hexadecimal command and read response.");
  Serial.println("Enter '3' to set up cycling through commands.");
  String cmd = Serial.readStringUntil('\n');
  if (String(cmd) == "0") {
    MyReset();
  }
  else if (String(cmd) == "1") {
    Serial.println("You can now send an SPI command.");
    Serial.println("Please enter a hexadecimal value that is 4 digits long (e.g., '4FA1').");
    MyCommand();
  }
  else if (String(cmd) == "2") {
    Serial.println("You can now send an SPI command.");
    Serial.println("Please enter a hexadecimal value that is 4 digits long (e.g., '4FA1').");
    MyCommandWithResponse();
  }
  else if (String(cmd) == "3") {
    Serial.println("This process will ask you to input a low hexadecimal value, a high hexadecimal value,");
    Serial.println("and the speed to cycle through all commands between them, inclusive.");
    Serial.println("You can now write the low value you wish to use.");
    Serial.println("Please enter a hexadecimal value that is 4 digits long (e.g., '4FA1').");
    SetCycleLowValue();
    Serial.println("You can now write the high value you wish to use.");
    Serial.println("Please enter a hexadecimal value that is 4 digits long (e.g., '4FA1').");
    SetCycleHighValue();
    Serial.println("You can now write the desired cycle speed in miliseconds.");
    Serial.println("Please enter a whole number of any size.");
    SetCycleSpeedValue();
    Serial.println("Start Cycling---");
    MyCyclingCommand();
  }
  delay(5000);
}
```