Digital Sampling Oscilloscope

Mary Anne Peters & Joseph Tylka
Department of Mechanical and Aerospace Engineering
Princeton University, Princeton, NJ 08544, USA

INTRODUCTION

This paper summarizes the construction and operation of a digital sampling oscilloscope, built from discrete analog and digital components, featuring an Arduino microcontoller. In Sec. I, we provide an overview of the primary components of the oscilloscope. We also provide some background and historical information on oscilloscopes in general. In Sec. II, we describe, in detail, the circuit theory behind each of the main components of the oscilloscope. We also discuss our specific implementation of these circuits in our oscilloscope, as well as the mechanisms we have implemented for user control, data retrieval, and waveform display. In Sec. III, we discuss the performance of the various components of our oscilloscope and display some captured data. Finally, in Sec. IV, we summarize the oscilloscope's performance, discuss some issues we faced in our construction process and the lessons learned. We include circuit diagrams of our oscilloscope as well as some of the larger figures in the Appendix. Also included in the Appendix are parts lists for our specific construction and the Arduino microcontroller code. We refer the interested reader to manufacturer datasheets[1] for each component in the parts list for additional information.

I. OVERVIEW

In this section we will give an overview of the digital sampling oscilloscope's primary functions, as well as an overview of the layout of the paper. The purpose of the digital sampling oscilloscope is to capture a voltage signal and save it to memory in order to later retrieve and display the signal. An oscilloscope is advantageous over a voltmeter because it extracts and displays high time resolution waveforms of oscillating (AC) and transient signals over a given time interval rather than providing a single time-averaged measurement.

The input stage of the oscilloscope consists of a variable DC voltage offset control and a variable gain control which allow the user to best display the captured signal. Since the analog to digital conversion (ADC) process takes a finite amount of time to complete, the signal is fed through a sample and hold circuit prior to the ADC chip. This step ensures that the voltage at the input of the ADC is constant over the duration of the conversion process. The digital representation of the sampled signal is then sent through a buffer to a static RAM (SRAM) chip. The data can then be read out through two digital to analog converters (DACs), one of which converts the SRAM addresses to provide a time scale, while the other converts the data stored in the SRAM to recreate the voltage signal. The analog outputs of the two DACs are sent to an analog display which plots the signals against each other. The data from the SRAM is also read by an Arduino Nano microcontroller and sent to a desktop computer via USB to be displayed in National Instruments' LabView. Figure 1 shows the main components of the oscilloscope as well as the user inputs, propagation of information and control signals throughout the system.

A. Background

Before discussing the details of our project it is valuable to briefly examine the history of oscilloscopes. Andre-Eugene Blondel invented the electromagnetic oscillograph in 1893 which used a galvanometer to trace a pen across a roll of paper to capture a waveform.[2] These first oscillographs were limited to a frequency range of 10 to 19 Hz due to the mechanical recording limitations (*i.e.* how fast the pen could move across the paper). The light-beam oscilloscope used a mirror and photographic plates to record the data improved upon this design. It could capture higher frequency signals up to about 500 Hz.

Ferdinand Braun invented the cathode ray tube (CRT) oscillograph in 1897. In the late 1930s, the company A. C. Cossor designed a dual-beam oscilloscope. It applied an oscillating sawtooth reference signal to horizontal deflector plates and the measured input signal to vertical deflector plates, creating a "sweep" of the input signal as

[1] Most datasheets can be found by simply performing a Google search for the part numbers listed in Tab. IV in the Appendix.

[2] Pereira, J.M.D. *The History and Technology of Oscilloscopes*, IEEE Instru. & Meas. Mag., vol. 9, pp. 27-35, 2006

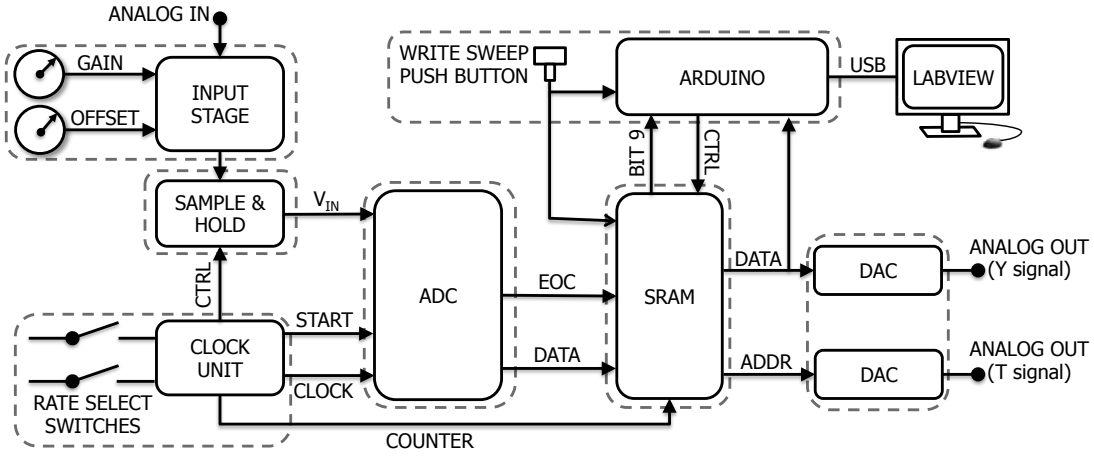


Fig. 1. Block diagram illustrating the primary components of the sampling oscilloscope. Boxes with solid lines correspond to the one or more circuit components shown in each OrCAD schematic in the Appendix. Boxes with dashed lines may include user inputs (*i.e.* the gain and offset knobs, the sampling frequency rate selection switches, and the write sweep push button) to indicate the contents of each individual OrCAD drawing.

a function of time on a phosphor display screen. However, this oscilloscope still had issues with drift because there was no fixed reference point for synchronizing the horizontal and vertical signals. In order to obtain a steady, repeating the signal on the display, the user had to tune the reference sawtooth signal until the signal no longer drifted across the display. In 1946 this problem was addressed by Howard Vollum's and Jack Murdock's invention of the triggered oscilloscope, which synchronizes of oscillatory waveforms by triggering at a given point on the signal. The Tektronix foundation (founded by Vollum and Murdock) refined this design for the commercial market and became the first manufacturer of the calibrated oscilloscope.

In 1981, Walter LeCroy filed a patent for the first digital oscilloscope.[3] This eliminated the necessity for horizontal and vertical plates to be used to display a sweep in the analog CRT oscilloscope. A digital oscilloscope uses an ADC to convert the analog inputs to digital signals, and saves those signals to memory to be displayed on a digital screen (*e.g.* an LCD display). Modern oscilloscopes feature very high input impedance (typically ~ 1 MΩ) to effectively isolate the circuit being measured from the oscilloscope, and are able to measure signals with frequency content up to ~ 10 GHz. However, this bandwidth is often limited by the capacitance of the cables used to probe the circuit. Essentially, the cable acts like a low pass filter since the conductors have some series resistance and capacitance to ground. Another common feature of modern oscilloscopes is a "scope probe" with an adjustable input impedance allowing the user two choose between an input impedance of $1\times$ or $10\times$ that of the oscilloscope. This probe is designed to have very little capacitance to ground, to allow a higher frequency range than typical coaxial BNC cables. Modern day instruments also have the ability isolate the AC component of a signal, by setting the oscilloscope to the "AC coupling" mode. This is particularly useful when the input signal has a large offset, but the user is only interested in rapid fluctuations in the signal. However, this removal of the DC offset is often achieved by simply introducing a "blocking" series capacitor, which will tend to filter out low frequency signals in addition to removing the DC offset. Alternatively, oscilloscopes have a DC coupling mode, which does not employ a series capacitor. Thus to maximize the bandwidth of the oscilloscope, users typically utilize the scope probe on the DC coupling mode. The oscilloscope we have built and will discuss in this report is a digital sampling oscilloscope that operates on the same principles at the one patented by LeCroy in 1981.

[3]LeCroy, W.O. *Test Probe*, US Patent 4423373, Filed March 16, 1981, Issued December 27, 1983

II. Oscilloscope Subsystems

In this section we will discuss the circuit theory and implementation of each of the oscilloscope's subsystems, as well as the principles of operation and implementation of external systems such as the Arduino and LabView. Each of the following subsections details the function and operation of a subsystem of the oscilloscope, beginning with the input stage. The final paragraph in each subsection summarizes the various inputs and outputs of each subsystem, as well as any mechanisms for user input.

A. Input Stage

The input stage of the oscilloscope consists of components which allow the user to adjust the gain and DC offset of the input signal. The analog input signal is sent through a series of three operational amplifiers (op-amps), as shown in Fig. 14 in the Appendix. The power rails of the op-amps are connected to ± 12 V, meaning the voltage range throughout the entire input stage is ± 12 V. However, as we will see in Sec. II-C, the sample and hold circuit will clip any signals outside of the range ± 5 V. Thus, our oscilloscope is capable of accepting signals up to a maximum amplitude of ± 12 V, but must fit within the range of ± 5 V at the output of the input stage by removing any DC offset, applying a gain of < 1, or both.

The first op-amp is wired in a voltage follower configuration. This configuration uses feedback to ensure that the output of the op-amp matches its input ($V_{out} = V_{in}$). Essentially, the op-amp works to minimize the voltage difference between its inverting and non-inverting inputs, thus reproducing the input at the output. This step serves to increase the input impedance of the entire oscilloscope to that of the op-amp (~ 1 MΩ), which will effectively isolate our oscilloscope from whatever circuit we connect to the input. That is to say, the oscilloscope does not draw very much current from the external circuit, so the behavior of that circuit will remain essentially unchanged. This is the benefit of high input impedances on voltage measuring devices.

The output of the voltage follower is sent into a second op-amp, configured as a linear inverting summing amplifier. This circuit creates a variable DC offset, which is controlled by adjusting the position of the knob (*i.e* the wiper) of a potentiometer whose other terminals are connected to ± 5 V, allowing for a DC offset of up to ± 5 V. In our case, we use a 100 kΩ potentiometer, but this value is not critical, provided that the power dissipated by the potentiometer ($P = \Delta V^2 / R_{pot}$) is reasonable (typically $< 1/2$ W). The output of the voltage follower and the wiper are connected to the inverting input of the second op-amp through individual resistors. The output of the op-amp is fed back to the inverting input and the non-inverting input is grounded. Thus, the voltage at the output of the summing amplifier can be calculated using the two ideal op-amp "golden rules":

1) The voltage difference between the inputs is zero.
2) The current flow into (or out of) each input is zero.

Employing these rules requires that all current flowing from the potentiometer and the voltage follower must pass through the feedback resistor. Also, the voltage at the inverting input must be zero.[4] Carrying out the circuit analysis using Kirchhoff's laws results in the expression given in Eq. (1), shown below.

$$V_{out} = -R_f \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} \right) \tag{1}$$

In this equation, R_f refers to the feedback resistance and R_1 and R_2 refer to the resistances connecting the inverting input to the signals V_1 and V_2, respectively. In our case, the three resistors for the summing amplifier all have the same resistance (10 kΩ), so $R_1 = R_2 = R_f$ and thus the overall gain is simply -1 (inverting). The output of the voltage follower (*i.e.* the input signal) is V_1 and the DC voltage at the wiper of the potentiometer is V_2.

The third op-amp is configured as a linear inverting amplifier used for variable gain amplification. Again employing the ideal op-amp rules and Kirchhoff's laws, we calculate the gain of the inverting amplifier, which is given by Eq. (2), shown below.

$$V_{out} = -\frac{R_f}{R_{in}} V_{in} \tag{2}$$

In this equation, R_{in} is the resistance connecting the inverting input to the signal V_{in}. In our case R_f is a variable resistor with $0 < R_f < 10$ kΩ and $R_{in} = 470\ \Omega$, thus we can apply a maximum gain of approximately $20\times$ to the input signal. We see again that this amplifier inverts the signal, resulting in a non-inverted signal at the output of the third op-amp, hence there is no need for an additional inverting amplifier. Table I summarizes the voltage range and gain characteristics of the input stage.

TABLE I
INPUT STAGE CHARACTERISTICS

Parameter	Value
Input Signal Range	± 12 V
Signal Range of S/H	± 5 V
DC Offset Range	± 5 V
Maximum Gain	$20\times$
Output Signal Range	± 5 V

We note that the variable offset and gain circuits could have been combined into a single op-amp circuit, simply by making the feedback resistance of the summing amplifier adjustable. However, we intentionally implement two separate op-amps to allow for more independent control for both the offset and gain. Specifically, the offset control is completely independent of the gain, yet the gain control will amplify any DC offset as well

[4]The op-amp creates what is known as a "virtual ground" at the inverting input.

TABLE II
CLOCK UNIT RATE SELECTION

Rate Setting	ADC Clock	Sampling Frequency
$\times 1$	7.2 kHz	150 Hz
$\times 8$	57.6 kHz	1.2 kHz
$\times 16$	115.2 kHz	2.4 kHz
$\times 64$	460.8 kHz	9.6 kHz

as the signal. We also note that a more complicated circuit could have been used to implement completely independent gain and offset. This circuit would require an initial variable DC offset summing circuit, to remove any DC offset from the original signal,[5] a variable gain linear amplifier to adjust the gain only on the AC component of the signal, and finally a second variable DC offset summing circuit, to add any desired offset.

To summarize, the input stage receives an analog input signal, and features two control knobs, allowing the user to adjust the DC offset and gain on the signal. The output of the input stage is sent to the input of the sample and hold circuit, to be discussed in Sec. II-C. First, we discuss the clock unit, which will also play a role in the sample and hold circuit.

B. Clock Unit

The clock unit of the oscilloscope is responsible for providing the relevant clocks to the digital circuits as well as necessary control signals. The main clock signals are provided by a bit rate generator which divides down the frequency of the signal from a crystal oscillator to generate many different frequency clock signals. In our circuit, two clock signals are taken from the bit rate generator, the first of which is used as the clock for the ADC chip, while the other sets the sampling frequency. The frequency of the ADC clock is chosen to be $48\times$ times the sampling frequency, to allow adequate time for the analog to digital conversion process to complete. A useful feature of the bit rate generator is that it has four different rate selection modes, allowing the oscilloscope to operate at any of four different sampling frequencies. The four possible clock frequency pairs are shown in Tab. II. See Fig. 15 in the Appendix for the schematic of the clock and control unit circuit.

First, our circuit synchronizes the clocks by feeding them into an edge-triggered D-type flip-flop, for which the data (D) input is the sampling frequency clock and the clock input is the ADC clock. Simultaneously, the ADC clock is sent to both the clock input of a 4-bit counter and, of course, the ADC. The output of the first flip-flop, denoted Q_1, is sent to the $\overline{CLR}$ input of the counter, to ensure that the counter is synchronized with Q_1. This means that the counter only counts for the first 24 ADC clock pulses, and the outputs are set to logical low (ground) for the next 24. The synchronized sampling frequency clock (Q_1) is also used to count through addresses of the SRAM, as described in Sec. II-E.

The input bits of the counter are all set to low and the outputs are used to create the desired control signals. We denote the output bits of the counter, from least significant to most significant, QA, QB, QC, and QD. Sending the signal $\overline{QA \cdot QC}$ to the $\overline{LOAD}$ input causes the counter to reset to all low outputs every 6 clock pulses. This signal is also used to clock a second D-type flip-flop, for which the data input is Q_1. The output of the second flip-flop, denoted Q_2, is a signal which is low for 6 ADC clock periods, high ($+5$ V) for the next 18, and then low again for the next 24, and repeats. Note that we refer to Q_1 and Q_2 as the outputs of the first and second D-type flip-flop *chips*, respectively, not to be confused with the outputs of the first and second flip-flop *circuits* on a single chip, as seen in Fig. 15. The use of two flip-flop chips is necessary since each chip uses a single clock for all of its flip-flops.

The signal $Q_1 \cdot \overline{Q_2}$ creates a "window" pulse 6 ADC clock periods long, which we use to gate the control signals. This signal is sent simultaneously to two "AND" gates. The first "AND" gate combines this signal with $\overline{QC}$ to create a 4 ADC clock period long pulse, which is used by the sample and hold circuit to sample the incoming analog signal. The second "AND" gate uses QC to create a 2 ADC clock period long pulse, immediately after the first pulse, which is used as the "start conversion" signal for the ADC. All of the timings of the signals described above are depicted in Fig. 2. The last signal shown in the figure is the "end of conversion" signal being created by the ADC, which will be discussed in Sec. II-D.

We note that the operation of this circuit is only possible due to the propagation delay inherent to the digital logic gates. The $\overline{QA \cdot QC}$ signal (used to clock the second flip-flop) has a rising edge slightly after the synchronized sampling frequency clock has its falling edge. Without the propagation delay, Q_2 would remain high indefinitely, since the only rising edges arriving to the clock input of the flip-flop would occur while the data input (Q_1) is high. This would eliminate the window pulse, meaning no control signals would be sent through their "AND" gates.

To summarize, the clock unit generates four signals

[5]Alternatively, a series capacitor could be used to block the DC component, but this would result in an attenuation of low frequency signals as well.

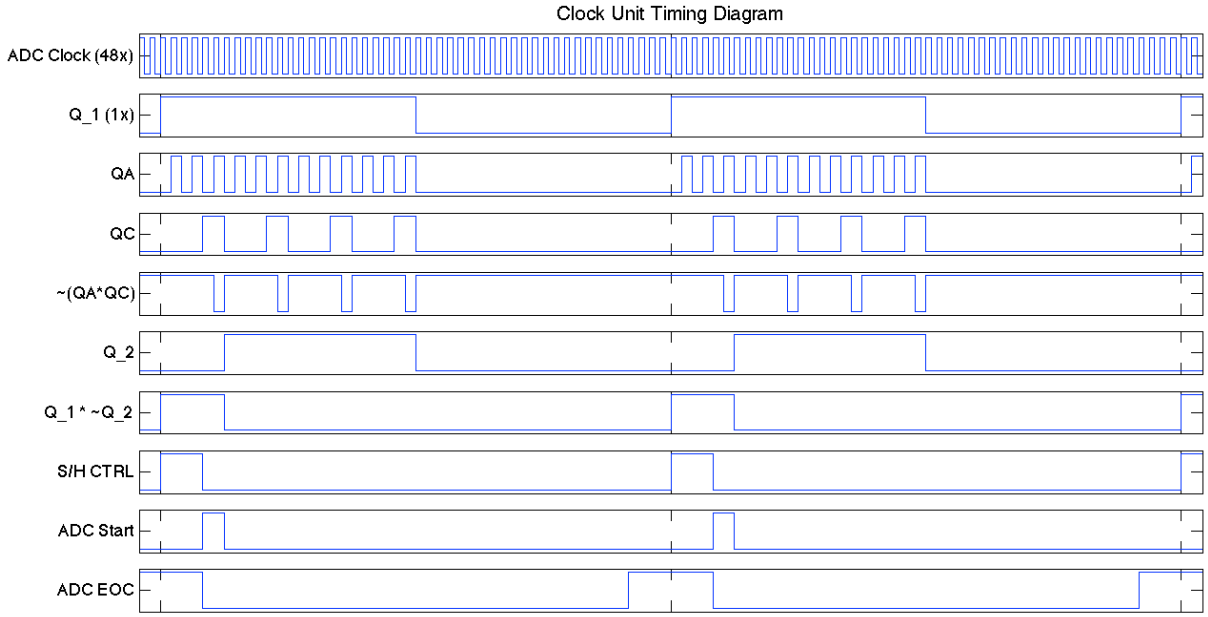


Fig. 2. Simulated timing diagram showing the relative timings of various signals in the clock and control unit.

that are used elsewhere in the oscilloscope. One such signal is the ADC clock, used to clock the ADC, and chosen to have a frequency $48\times$ times that of the sampling frequency clock. The sampling frequency clock is also generated by the clock unit, and is used to count through the addresses of the SRAM. The clock unit also generates two consecutive control pulses at the start of each sampling frequency clock period. The first pulse has a width equal to 4 ADC clock periods and is used to control the sample and hold circuit, while the second pulse has a width equal to 2 ADC clock periods and is used to start the analog to digital conversion process. The user is able to select the sampling frequency with two digital switches, yielding four different frequencies.

C. Sample and Hold

The sample and hold circuit is necessary to ensure that the voltage at the input to the ADC is held constant over the duration of the conversion process, as any fluctuations may result in errors in the conversion. The circuit consists of two voltage followers, a bilateral CMOS transistor switch, and a capacitor, as shown in Fig. 16 in the Appendix. The op-amps are again powered from ± 12 V, and the bilateral switch is powered with $V_{DD} = +5$ V and $V_{SS} = -5$ V. The signal from the input stage passes through the first voltage follower and is sent to the input of the bilateral switch. Thus any signal outside of the range ± 5 V will be clipped by the bilateral switch. The voltage follower is necessary to isolate the bilateral switch from the input stage, so that currents from the input stage are not run directly through the switch.

The output of the bilateral switch is connected by a capacitor to ground and also to the input of the second voltage follower. Thus, when the switch is closed, the first voltage follower charges the capacitor until the voltage across it matches the voltage at the non-inverting input of the first op-amp. This process is known as the "sampling" phase, since the voltage across the capacitor is continually being adjusted to match the input signal. When the switch is open, the capacitor maintains that voltage at the non-inverting input of the second op-amp. This state is known as the "holding" phase. The second voltage follower serves to isolate the capacitor from the subsequent electronics.

The bilateral switch is electronically controlled, meaning setting the control voltage equal to V_{DD} closes the switch and setting the control voltage equal to V_{SS} opens the switch. In fact, due to the nature of a transistor switch, any control voltage less than $V_{DD}-V_T$, where V_T is some threshold voltage (typically ~ 0.7 V), will open the switch. Therefore, we can use a TTL level signal ($+5$ V and 0 V), to control the switch. Thus by sending the control signal from the clock unit (discussed in Sec. II-B) to the control input of the bilateral switch, we sample the input signal for four ADC clock periods, and hold the

final sampled voltage for forty four periods. Figure 6 shows some examples of the output of the sample and hold circuit with a sinusoid input at various sampling frequencies.

Ideally, the capacitor would be able to maintain the held voltage constant indefinitely, but due to non-idealities such as bias currents through the op-amps (typically ~ 80 nA), the voltage can only be held for a short time. The voltage decay rate is given by the formula below.

$$\frac{dV_C}{dt} = \frac{i_{bias}}{C} \tag{3}$$

In this equation, V_C is the voltage across the capacitor and i_{bias} is the bias current through the op-amp. Clearly, increasing the capacitance C would decrease the voltage decay rate, but we must also consider the charging rate of the capacitor, whose time constant is given in the equation below.

$$\tau = (R_{switch} + R_o)C \tag{4}$$

In this equation, R_{switch} is the resistance of the bilateral switch in the closed position, typically $\sim 110\ \Omega$, and R_o is the output resistance of the op-amp, typically $\sim 75\ \Omega$. Thus, as expected, increasing the capacitance will increase the charging time. Hence, the capacitance value must be chosen to simultaneously optimize the charging speed and the hold time for a given sampling frequency.

In our case, we chose C to be 68 nF, which gives us a charging time constant of $\sim 13\ \mu$s. At an ADC clock speed of 57.6 kHz (the $\times 8$ setting), the capacitor is given $\sim 70\ \mu$s to charge, which, compared to the time constant, is plenty of time. Also, the capacitor holds the voltage for $\sim 770\ \mu$s. The voltage decay rate for this capacitance is ~ 1.18 V/s, yielding a total loss of ~ 0.9 mV over the entire hold period. As our input signal range is only ± 5 V, this loss corresponds to a $\sim 0.01\%$ error, which we can tolerate, bearing in mind that the calculations above are based on nominal values given in the datasheets.

When the bilateral switch is closed, the capacitor also acts like a low pass filter. The cutoff frequency of a low pass filter is specified as the frequency at which the amplitude of the input signal is attenuated by -3 dB. The formula for calculating the cutoff frequency is given the equation below.

$$f_c = \frac{1}{2\pi\tau} \tag{5}$$

Therefore, using the time constant calculated above, the cutoff frequency of our circuit should occur around ~ 12.5 kHz. We will see in Sec. III-A that the actual cutoff frequency of our circuit occurs around 16 kHz, which is easily within the tolerance of the nominal values we used in the above calculations.

To summarize, the sample and hold circuit receives the output of the input stage and the control signal from the clock unit as inputs, and outputs a sampled version of the input signal. This signal is then passed to the ADC, as will be discussed in the next subsection.

D. Analog to Digital Converter

The analog to digital converter (ADC) receives the output of the sample and hold circuit and calculates a binary number to represent that voltage. There are many methods that can be used to perform this calculation, which vary in speed, accuracy, and complexity. For example, a "direct conversion" or "flash" ADC is rather complex but has a very fast conversion time. This type of ADC consists of 2^N resistors and $2^N - 1$ comparators, where N is the number of output bits (binary digits). The ADC functions by feeding the input signal, V_{in}, to all of the comparators simultaneously, while the other inputs of the comparators are connected to a discretized range of reference voltages, V_{ref}. For example, an 8-bit flash ADC would have 256 resistors of equal resistance in series, with each node between resistors connecting to a different comparator's reference input. Essentially, the 255 comparators *compare* the input signal to all of the reference voltages and output a logical high (or "true") if $V_{in} > V_{ref}$ or a logical low ("false") if $V_{in} < V_{ref}$. The outputs of the comparators are sent to a logic circuit which (almost) instantaneously determines the digital output.

On the other hand, a ramp comparison ADC is very simple but can take a long time to convert. One implementation of this type of ADC uses an N-bit binary counter fed into an N-bit DAC (see Sec. II-G) to create a staircase-like ramp. The input signal is compared to the ramp signal using a comparator so that the instant when the output of the comparator changes corresponds to the ramp voltage crossing the input signal voltage. Thus the output bits of the counter at that instant are precisely the digital representation of the analog input signal. Clearly, this method will be much slower than the flash ADC, since this ADC must count through all of the possible digital values. Therefore the speed of the ADC depends on the clock speed of the counter, and the conversion time increases as the resolution (number of bits) increases.

In our oscilloscope, we use a clocked 8-bit successive approximation ADC, which systematically determines the 8-bit number which corresponds to the input signal voltage, V_{in}, by generating various reference voltages and comparing them to the input signal. To do this, the ADC uses a comparator, a resistor network, analog

switches, and control logic. The resistor network consists of 256 resistors of equal resistance in series, which divide the total voltage difference across the network into evenly spaced discrete voltages, each of which is sent to one end of an analog switch. The control logic closes one of the switches to send a certain voltage, V_{ref}, to the reference input of the comparator, while V_{in} is sent to the other input. The result of the comparison is used by the control logic to decide which switch to close next. By repeating this process several times, the ADC is able to determine which of the 256 reference voltages is nearest to that of the input signal, and represents that reference voltage with an 8-bit number.

Our ADC determines the 8-bit output in exactly 40 clock periods, which explains our choice for the ADC clock to be $48\times$ our sampling frequency. Also, as specified in the datasheet, the ADC requires a "start conversion" pulse between 1 and 3.5 clock periods long, which justifies our choice for a 2 clock period long pulse. This pulse tells the ADC to begin converting V_{in}, and causes the ADC's output called "end of conversion" (EOC) to transition to logical low. The EOC output remains low for the duration of the conversion process (40 clock periods) and then transitions to logical high to indicate, as the name suggests, the end of the conversion process. The EOC signal is simultaneously sent to the control logic of the SRAM (to be discussed in Sec. II-E) and fed back to the "output enable" (OE) input of the ADC. This input controls the data outputs on the ADC, where a logical low at the OE input prevents data from being transmitted out from the ADC and a logical high at the input allows transmission. See Fig. 17 in the Appendix for the schematic of the ADC.

The data outputs are what are known as "Tri-State"[6] outputs, indicating that the output pins are always in one of three possible states. When the OE input is high, *i.e.* the data outputs are *enabled*, the ADC sets the voltage at those pins to their designated voltages, either logical high or low, as determined by the conversion process. However, when the OE input is low, *i.e.* the outputs are *disabled*, the output pins may take on an intermediate voltage level or a level not determined by the ADC. The significance here is that the output pins are no longer "driven" to a particular value by the ADC itself, meaning the ADC is not attempting to control the voltages at the outputs. This allows those pins to be driven by other components. For example, the outputs of the ADC could be connected directly to the inputs of the SRAM, which are also its outputs. The Tri-State property of the ADC outputs ensures that if the ADC's output is disabled, those data pins can be driven by the SRAM without affecting the ADC.

This feature alone does not solve every problem. For example, if the ADC's outputs were enabled and the SRAM were outputting the saved data at a given address, both devices would be trying to drive their output pins to certain levels. Thus, if, on a certain pin, the two levels were conflicting (*e.g.* the ADC says high while the SRAM says low), we end up shorting one of the devices, and possibly damaging one or both of the chips. Therefore, care must be taken to ensure that the outputs of the ADC will never be in conflict with another device. We discuss the steps we have taken towards this goal in Sec. II-E.

To summarize, the ADC takes as inputs an analog voltage (V_{in}) from the sample and hold circuit, a clock signal from the clock unit, a start conversion pulse also from the clock unit, and an output enable (OE) signal which is fed back directly from the end of conversion (EOC) output. The ADC passes the 8-bit data values to the buffer for the SRAM and the EOC signal to the SRAM's control logic.

E. Data Storage (Buffer, Memory, and Address Counter)

The SRAM is responsible for storing a sequence of 8-bit values computed by the ADC, so that the sequence may be retrieved and displayed. The 8 digital signals being sent from the ADC are passed through digital buffers to the data pins of the SRAM to be stored to memory. The lowest 8 output bits of a 12-bit counter are connected to the lowest 8 address bits of the SRAM to count through the different addresses of the SRAM. By synchronizing the counter and the ADC, we can ensure that each 8-bit value is written to a single address in the SRAM before the counter moves on to the next address. The SRAM and buffers are controlled by a "NAND" gate to coordinate the transmission of data through the buffers and the writing and reading of data to and from the SRAM. The operation of the SRAM and the 12-bit counter is intimately related to the functions of the Arduino, which will be touched on here, and discussed in more detail in Sec. II-F. The schematic for the SRAM, the buffers and the 12-bit counter is shown in Fig. 18 in the Appendix. Note that the data input pins of the SRAM are also its data output pins.

As mentioned above, the data outputs of the ADC are connected to the inputs of eight digital buffers. These buffers are controlled such that when the voltages at the gate pins (denoted by $\overline{OE}$ in Fig. 18) are low, the outputs are *enabled*, meaning the buffers drive their outputs to

[6]Tri-State is a registered trademark of National Semiconductor Corp.

TABLE III
SRAM CONTROL TRUTH TABLE

$\overline{CS}$	$\overline{WE}$	$\overline{OE}$	Operation Mode
High	×	×	Not Selected
Low	Low	×	Write
Low	High	Low	Read
Low	High	High	Inactive

match their respective inputs. If the voltages at the gate pins are high, the outputs are *disabled*, and are said to be in a "high impedance" state, in that they are essentially treated as open circuits. Our buffer chip has two gate pins, each controlling four of the eight buffers. However, we send the same signal to both gate pins so all eight buffers are controlled simultaneously. The control logic that we have implemented controls both the gates on the buffers as well as the write enable pin on the SRAM, so we will discuss the operation of the SRAM first.

The SRAM has three control inputs which are used to put the SRAM in one of three operational modes. The three inputs are chip select ($\overline{CS}$), write enable ($\overline{WE}$, and output enable ($\overline{OE}$). Note that each of these pins are noted with a bar over the letters, signifying that a logical low level at the input "activates" that function. For example, the $\overline{CS}$ input requires a low level in order to "select" (or enable) the chip. If a high level is sent to the $\overline{CS}$ input, the chip cannot be used. In our case, we have connected the $\overline{CS}$ input directly to ground, so that our chip is always selected.

Provided that the chip is selected (*i.e.* $\overline{CS}$ is low), the $\overline{WE}$ determines if the chip is in read mode or write mode. When the $\overline{WE}$ input is low, the $\overline{OE}$ input is ignored, and the chip is in write mode. In this mode, any signals arriving to the data inputs will be stored to memory at the address specified by the signals at the address pins. When the $\overline{WE}$ input is high, the chip may either read out the data, or do nothing, depending on the state of the $\overline{OE}$ input.

Provided that the chip is selected and is not in write mode (*i.e.* $\overline{WE}$ is high), if the $\overline{OE}$ input is low, the outputs are enabled, and the SRAM reproduces, at the data outputs, the digital values saved in memory. These values are equal to those stored at the address specified by the signals at the address pins. If the $\overline{OE}$ input is high, then the chip is essentially inactive, since it is in neither read nor write mode. The truth table describing the different operational modes of the SRAM is given in Tab. III.

The control logic for both the gates of the buffers and the write enable input of the SRAM is a "NAND" gate. The inputs to this "NAND" gate come from the ADC and the Arduino. Recall that a "NAND" gate produces a low voltage only if both inputs are high. If either input is low, the other input is irrelevant and the output of the "NAND" gate must be high. The signal from the Arduino is high for exactly the time it takes for the counter to cycle through all of the addresses[7] in the SRAM. The other input of the "NAND" gate is the end of conversion (EOC) signal from the ADC, which goes high as soon as the conversion is complete, and returns low again at the start of the next conversion. Recall that this signal is fed back to the output enable (OE) input of the ADC, meaning that the data is only fed into the buffers during that gap between conversions. Therefore, when the Arduino signal (from pin D13, to be discussed in Sec. II-F) is high, the control signal from the "NAND" gate will go low for the exact same gap between conversions. Thus for that small window of time, the ADC's outputs are enabled, the buffers' outputs are enabled, and the SRAM is put into write mode. Conversely, for the entire duration of the conversion processes, the ADC's and buffers' outputs are disabled, and the SRAM is in read mode. We note that this causes the SRAM to essentially "hold" those values which were just written until the SRAM's address inputs are changed. However, when the D13 signal from the Arduino is low, no data can be written to SRAM, and the buffers' outputs are always disabled. Therefore, the outputs of the SRAM will continue to read out the values stored in memory at whatever addresses are being specified.

As mentioned above, the first 8 output bits of the counter are connected to the first 8 address bits on the SRAM. The other 3 address bits are all connected to ground. Therefore, we are essentially only using one eighth of our memory capacity. This choice is necessary since we use an 8-bit DAC to convert the outputs of the counter into a sawtooth wave. The purpose of this sawtooth wave will be explained in Sec. II-G. The counter is clocked by the sampling frequency clock generated by the clock unit, as discussed in Sec. II-B. The clock input is denoted by $\overline{CLK}$, indicating that the falling edge of the clock signal triggers the transition to the next value. As can be seen from the timing diagram in Fig. 2, the SRAM's address changes long after the data has been written to the previous address, which occurs during the high pulse of the EOC signal.

The reset input of the counter is connected to the write sweep push button, shown in Fig. 19 in the Appendix. This means that when the user presses the button, all

[7]Strictly speaking, we only use a subset of the possible SRAM addresses since we also use an 8-bit DAC to convert the addresses to an analog signal. See Sec. II-G for more details.

of the outputs of counter are set and held low. When the user releases the push button, the counter may begin to count through the addresses. The push button is also connected to the Arduino input D3 (see Sec. II-F), which monitors that pin for a rising edge. When a rising edge is detected, the Arduino changes its D13 output to high, indicating that the SRAM may be written to. The 9^{th} output bit of the counter, Q_9, is sent to the Arduino input D2, which also monitors that pin for a rising edge. For this pin, when a rising edge is detected, the Arduino changes its D13 output to low. Therefore, the counter begins to count up indefinitely as soon as the push button is released, but once Q_9 transitions from low to high, the D13 output of the Arduino becomes low, which prevents any more data from being written to the SRAM. In this way, the SRAM is guaranteed to only cycle through all of its addresses once without overwriting any stored data. We note that the counter simply continues to count through all of its possible values, and repeats once it reaches its maximum. Therefore, Q_9 will experience many rising edges, but this has been accounted for the in the Arduino code, as we will discuss in Sec. II-F.

The data inputs/outputs of the SRAM are connected both to the data inputs of an 8-bit DAC and to 8 digital inputs of the Arduino, to be discussed in Sec. II-G and Sec. II-F, respectively. The lowest 8 output bits of the counter are also connected to an 8-bit DAC (also in Sec. II-G). These connections will allow us multiple ways of displaying our captured waveforms.

To summarize, the digital outputs of the ADC are sent via the digital buffers into the data inputs of the SRAM. The data inputs of the SRAM are simultaneously connected to the data inputs of an 8-bit DAC and 8 digital inputs of the Arduino. The "NAND" gate receives an input from the Arduino's D13 output as well as the ADC's EOC signal. This control logic coordinates sending data through the buffers as well as writing that data to the SRAM. The 12-bit counter which counts through the addresses of the SRAM is clocked by the sampling frequency clock generated in the clock unit and is reset when the user presses the write sweep push button. The addresses of the SRAM are also sent to an 8-bit DAC, and the Q_9 output of the counter is sent to the Arduino.

F. Arduino Nano Microcontroller

As is evident in the previous section, the Arduino plays an integral role in the operation of our oscilloscope. It is responsible for initializing the procedure to fill the SRAM with data corresponding to one sweep of the input signal. Additionally, the Arduino reads the data that is present at the data input/output pins of the SRAM and sends those 8 bits as a byte across a serial line to the computer. The data can then be viewed in LabView, as will be discussed in Sec. II-H. The write sweep push button is connected to the Arduino, which the user presses to start the writing process. The schematic for the Arduino is shown in Fig. 19.

Before discussing our specific Arduino code, we will discuss the Arduino programming environment and code structure in general. The Arduino programming environment is very similar to C and C++, but with many unique built-in functions. The primary purpose of the Arduino is to repeatedly execute the code within the main loop, called with `void loop()`, while the Arduino is powered on. Of course, as with any other programming environment, some initialization steps must be taken. The very first lines of code are the variable declaration and initialization commands.

Immediately following the variable declarations is the setup routine, called with `void setup()`. Many important actions are performed inside this function. For example, the programmer may specify which of the digital pins on the Arduino will be used as digital inputs (reading digital signals) and which will be digital outputs (producing digital signals). Similarly, the analog pins can be declared as analog inputs (reading analog signals through and ADC) or analog outputs (producing pulse-width modulated analog signals). Also, the communication settings on the Arduino are declared in the setup function, such as initializing a serial connection and specifying its baud rate.

A useful feature of the Arduino are its "interrupt" pins. These pins may function as regular digital pins, or as interrupts, which are given special priority in the Arduino's processing. To function as an interrupt, the interrupt pin must be configured to detect a certain signal feature, such as a low to high transition (using `RISING`), a high to low transition (using `FALLING`), or either transition (using `CHANGE`). The interrupts may also be configured to detect when the signal is either high or low using `HIGH` or `LOW`, respectively. When the specified feature is detected, the Arduino immediately abandons whatever processing it was running, and executes the interrupt's designated function. Once the interrupt task is completed, the Arduino returns to the main loop. Note that the interrupts are given numbers (typically 0 and 1) which may differ from the digital pins on which they are enabled (typically 2 and 3, respectively).

Digital pin D13 is unique in that it is connected to an LED on the Arduino's circuit board in addition to functioning as a regular digital pin. This makes pin D13 especially useful for debugging, as visual feedback can

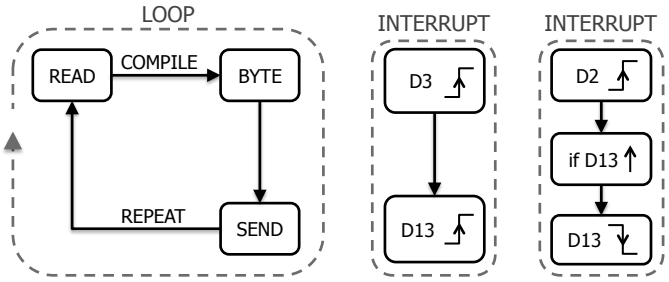


Fig. 3. Block diagram of Arduino code. Boxes with dashed lines indicate independent functions within the code.

be given the user immediately. Once the Arduino code has been compiled, it must be uploaded to the device via USB. All subsequent communication between the Arduino and the PC (such as data being sent to the serial port) also takes place across the USB.

We will now discuss the control code we have implemented in the Arduino. The pseudocode version of the code is given in Alg. 1 and the actual code is reproduced in Fig. 9 in the Appendix. Line numbers given in this section refer to the pseudocode in Alg. 1. Also, Fig. 3 depicts the structure of the Arduino code as a block diagram.

Algorithm 1 Arduino Control Code

```
1:  loop
2:      Read inputs D5 through D12
3:      Compile data into byte
4:      Send byte across the serial line
5:  end loop
6:  interrupt (D3: low → high)
7:      Set D13 = high
8:  end interrupt
9:  interrupt (D2: low → high)
10:     if D13 == high then
11:         Set D13 = low
12:     end if
13: end interrupt
```

For capturing data with the Arduino, we have 8 digital pins, D5 through D12, configured as digital inputs. The primary loop of the Arduino begins by reading the signals at those digital inputs (line 2). The Arduino then compiles those 8 bits of data into a single byte (line 3) which is sent over the serial line to the PC (line 4). This data can then be viewed using LabView, as we will discuss in Sec. II-H.

We have two digital pins, D2 and D3, which are configured as digital inputs and are used as interrupts. Both interrupts are set to monitor their respective inputs for a rising edge, *i.e.* a low to high transition (lines 6 and 9). We also have the D13 digital pin configure as a digital output. The write sweep push button is connected to D3, so when the user presses the button, the Arduino immediately executes the function specified as the interrupt response. In our case, the function that we execute sets the D13 pin to high (line 7), which, as discussed in Sec. II-E, enables the SRAM control logic so that data may be written to memory. Also from Sec. II-E, the Q_9 output of the counter is sent to D2, so when the last address is reached by the counter and Q_9 goes high, the second interrupt function is executed. In our case, the function begins by checking the state of D13 (line 10) and, if D13 is currently high, the function sets it to be low[8] (line 11). When D13 returns low, data may no longer be written to memory.

We note that, regardless of the interrupt functions, the Arduino is always streaming the data from the outputs of the SRAM to the PC. This means that even when data is no longer being written to memory, the Arduino will continue to stream the same waveform to the PC, since the counter will continue to cycle through addresses and the SRAM will be locked in read mode until the user presses the write sweep push button.

To summarize, the Arduino has two interrupt pins, D2 and D3, which monitor the Q_9 output of the 12-bit counter and the write sweep push button, respectively, for rising edges. The output D13 sends a control signal to the SRAM's control logic to indicate that data may be written to the SRAM. Also, the data pins on the SRAM are connected to 8 digital inputs on the Arduino, D5 through D12, so that the Arduino may send that data to the PC. The write sweep push button essentially starts a chain reaction of events to facilitate the writing of a single sweep of the input signal to memory. First, the 12-bit counter is cleared, so that the SRAM will begin writing at the very first address. Simultaneously, the D13 output of the Arduino is set high, so that the SRAM's control logic may allow data to be sent through the buffers and written to memory. When the user releases the push button, the counter begins to count and the ADC begins filling the memory. Exactly 256 sampling frequency clock periods after the push button is released, Q_9 goes high and the D13 output is set low, so that data may no longer be written to memory.

G. Analog Display

The digital to analog converters (DACs) that we use allow us to view the sweep of the input signal that is being written to, or already stored in the SRAM on an

[8]Without that "if" statement, the Arduino would waste processing time by setting the D13 pin low when it is already low.

analog display. As mentioned previously, we have two DACs, one which converts the addresses being sent to the SRAM to create a sawtooth wave, while the other converts the data saved in memory at those addresses. The schematic for the wiring of the two DACs is shown in Fig. 20 in the Appendix, and the circuit design can be found in the "Typical Application" section of the datasheet[9] with power and reference voltages changed from ± 10 V to ± 5 V. Note that each of our DACs are wired in the exact same configuration, with the only differences being the data bits coming in, and therefore the analog signals coming out.

The DAC receives 8 bits of data and determines, in approximately 100 ns, the appropriate output relative to its reference voltages. In our case, we give the DAC references to $+5$ V and ground (inputs V_{R+} and V_{R-}, respectively), and the DAC assumes symmetry about the ground reference. Thus the DAC produces -5 V when all inputs are low, and $+5$ V when all inputs are high. Strictly speaking, the DAC operates using a reference *current*, and drawing relative amounts of current from the outputs. For example, the $+5$ V reference voltage is connected to the DAC through a 5.1 kΩ resistor. This corresponds to a reference current of $\sim$ 1 mA flowing into the DAC from V_{R+}. Also, the outputs of the DAC are connected through 10 kΩ resistors to $+5$ V. Therefore, when all of the data inputs are high, the DAC draws ~ 1 mA from the I_{OUT} output while drawing no current from the $\overline{I_{OUT}}$ output. Conversely, when all data inputs are low the DAC draws no current from the I_{OUT} output while drawing ~ 1 mA from the $\overline{I_{OUT}}$ output. In both cases, the current flows through a 10 kΩ resistor, yielding either a voltage drop of 10 V from 1 mA, or no voltage drop. Therefore, the voltage at the $\overline{I_{OUT}}$ output is exactly the digital input values mapped to an analog voltage range of ± 5 V. The voltage at the I_{OUT} output is the same signal but inverted (*i.e.* a gain of -1).

To display the outputs of the DACs, we send the converted addresses to channel 1 of a commercially available, professionally built oscilloscope[10] and we send the converted data to channel 2. The professional oscilloscope has the ability to plot these two functions against each other, known as "X-Y mode", which produces the sweep as a function of address. The result is exactly the sweep of the input signal that we have captured and stored to memory being displayed on the screen of the professional oscilloscope. We emphasize that the purpose of using a professional oscilloscope is simply for the ease of access to an LCD display. Figure 8 shows the output of each DAC as well as the X-Y mode output. We refer to the analog signal produced by converting the addresses as the "T" signal, indicating that this sawtooth wave acts as our time base, and we refer to the analog signal produced by converting the data as the "Y" signal, since it is exactly the captured waveform. We note that our T signal exhibits some flattening near the ends of the diagonal ramp, which will be discussed in Sec. III-A.

As discussed in Sec. I-A, the professional oscilloscope generates its own internal "sweep" to display the incoming signals on the screen. In its standard operational mode, known as "Y-T mode", each channel is plotted as an independent function of time. However, the oscilloscope triggers its internal sweep by setting a threshold on one of the channels. Hence when the incoming signal crosses the trigger threshold, the signal is swept across the screen. An oscillating signal may cross the threshold many times, and thus if we have a repeating portion of a signal which does not match up exactly at the start and end of the sweep, the result on the screen is the same signal being overlaid many times, but starting at different points in the sweep. In our case, the signal being sent to the professional oscilloscope is exactly a repeating portion of the input signal, since we are reading through the SRAM continuously. Of course, our sweep of the input signal need not match up at the beginning and the end, so we will have discontinuities in the signal. This behavior can be seen in the top panels of Figs. 12 and 13 in the Appendix.

In X-Y mode, this is not the case. The T signal that we are providing is used as the sweep for the display, meaning a negative voltage (*e.g.* -5 V) would correspond to the left half of the time axis, while a positive voltage (*e.g.* $+5$ V) would correspond to the right half of the time axis. Essentially, the voltage of the T signal maps directly to the horizontal position of the signal seen on the screen. Simultaneously, the voltage of the Y signal directly controls the vertical position of the signal on the screen.

To summarize, the DACs take as inputs the 8-bit address used for the SRAM and the 8-bit values stored at those addresses, and produce two analog signals, T and Y, respectively. The T signal is simply a sawtooth wave corresponding to a linear progression through all of the addresses. The Y signal is exactly the portion of the input signal we have stored in memory.

H. LabView

Once the byte of data has been sent from the Arduino to the PC via USB, we display the data using National

[9]Texas Instruments DAC0800 Datasheet, June 1999, revised February 2013.

[10]We used the Tektronix TDS 210.

Instruments' LabView program. The overall goal of the program is to plot the data on a graph in real-time. To do this, we use the Arduino as a serial device (see Sec. II-F) which communicates through a COM port on the PC. We refer the interested reader to other references on serial port devices and communication for a more detailed explanation on their operation and functionality. LabView monitors the COM port for data, and converts the bytes it receives to decimal, to be plotted as a function of time.

The two main components of all LabView programs are the block diagram and the front panel. The block diagram is essentially the programming environment for LabView, where the program's instructions are defined. The front panel is essentially the graphical user interface (GUI) that is displayed while the program is running. Many blocks in the block diagram correspond exactly to elements on the front panel, which the user may interact with.

Fig. 10 in the Appendix shows the block diagram for our LabView program and the front panel. The blocks on the left side of the block diagram (external to the thick black box) are responsible for retrieving the data from the serial port and passing it elsewhere in the program. The pink, left most block titled "Serial Port Settings" corresponds to a selection of drop-down boxes on the front panel, where the user may specify the serial port settings. The information is passed from the serial port settings block and separated by the yellow block to its immediate right. This block passes the serial port settings to a LabView VISA[11] block which communicates directly with the COM port, using the port settings specified by the previous blocks.

The VISA passes the bytes of data arriving at the serial port and passes them to the instrument block (denoted "Instr") inside the outer of the two black boxes. The outer of the two black boxes is a programming loop, while the inner of the two is a case structure. The loop runs continuously while the program is running, and the case structure checks a condition in order for its contents to be evaluated. Our case structure is checking to see if the number of bytes at the serial port is greater than zero. If the number of bytes at the serial port is greater than zero, three events are triggered. First, the bytes of data are converted to ASCII characters, and sent to the block titled "RAW ASCII". These ASCII characters are displayed on the front panel when the program is running. We note that since we are sending bytes of data that correspond to a waveform, the ASCII characters are essentially meaningless. However, the user will be able to see the characters change if data is indeed flowing to the serial port. Simultaneously, the bytes of data are converted to decimal and sent to "Waveform Chart", which displays the real time values of the data on the front panel. The "Waveform Chart" block is located outside of the case structure so that chart will continue plotting, even if no new data is arriving at the port. Finally, the number of bytes at the port are plotted on "Waveform Chart 2", thereby displaying the real time data rate through the serial port. This chart is also shown on the front panel. The loop also has an "Iterations" counter, which, like the RAW ASCII block, serves as a debugging tool, since the counter will continuously increase if the loop is running. The pseudocode for this LabView program is given in Alg. 2.

[11]Visit the National Instruments website for more information on VISA structures.

Algorithm 2 LabView Code

```
loop
    if Bytes at Port > 0 then
        Convert data to ASCII and decimal
        Display ASCII data
        Plot data rate
    end if
    Plot decimal data
    Increment counter
end loop
```

To summarize, LabView receives a stream of bytes of data from the Arduino via the USB COM port and displays the decimal equivalents of that data on a graph in real-time. LabView also displays its incoming data rate as a function of time. The user may adjust the serial port settings in LabView to match those set by the Arduino.

III. Results

In this section we present collected data from our oscilloscope and measurements of the oscilloscope's performance. The first set of measurements discussed in Sec. III-A were collected using a professional bench top oscilloscope, captured with Tektronix's WaveStar program, exported as .csv files and plotted in MATLAB. These signals were measured at the output of the sample and hold circuit, prior to digital conversion, and are intended to illustrate the capabilities and limitations of our oscilloscope. Thus, we have assumed that the analog to digital conversions will be completed accurately and that the resulting digital values will be stored to memory without errors. Unless otherwise stated, the input signal to the oscilloscope was a sine wave with frequency 72 Hz.

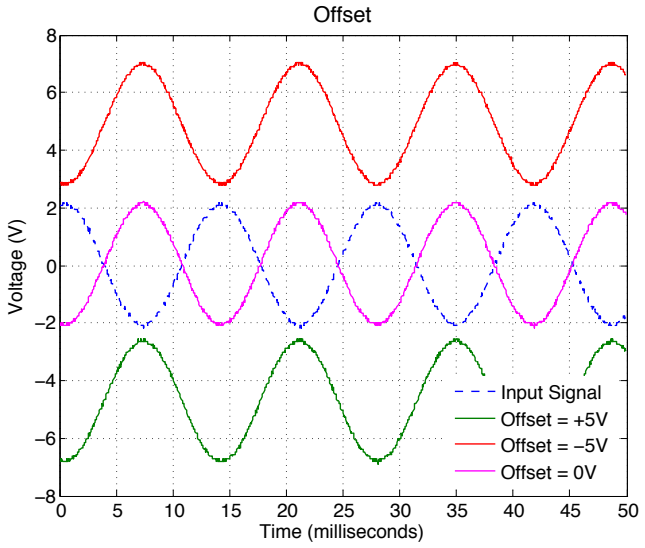


Fig. 4. Variable DC offset capabilities of the oscilloscope. The blue dotted line is the input signal, the solid lines correspond to an offset of -5 V (red line), 0 V (blue line) and $+5$ V (green line). The inversion of the signal is due to the measurement location and is not present in the final signal produced by the input stage.

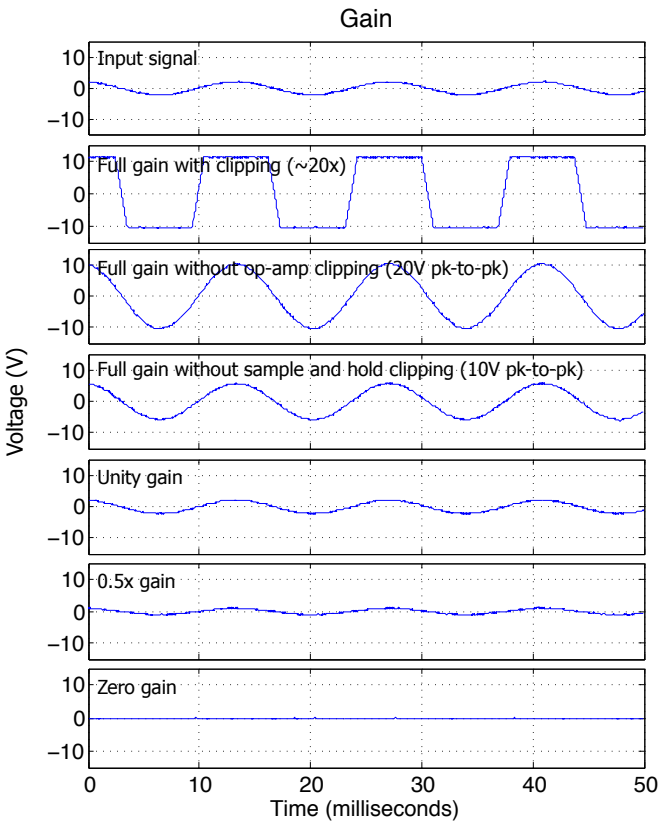


Fig. 5. Variable gain capabilities of the oscilloscope. The input signal is shown on the top panel. Panels two through six show the signal with various gain settings in decreasing order. The gain setting is given on the left side of each panel.

To demonstrate the end-to-end oscilloscope performance, we obtained oscilloscope captures with LabView via the Arduino along with the outputs of the DACs viewed on the professional oscilloscope for comparison. The LabView data is presented in Sec. III-B. Please note that many of the larger figures have been placed in the Appendix consolidate the bulk of the text.

A. Measured Performance and Limitations

Our oscilloscope has adjustable gain and offset as discussed in Sec. II-A. The output of the summing amplifier is shown for three different offset values (-5 V, 0 V and $+5$ V) in Fig. 4. These signals were obtained prior to the variable gain linear amplifier, so the signal is inverted relative to the input signal. This also means that an offset of $+5$ V shifts the signal down to -5 V, and *vice-versa*. This inversion is of course corrected by the variable gain amplifier as discussed before. Figure 4 displays the maximum ($+5$ V) and minimum (-5 V) range of the offset circuit. The noise added to the signal from the offset circuit is minimal ($< 1\%$).

Fig. 5 shows the output of the linear amplifier for several different gain settings compared to the input signal (shown in the top panel of Fig. 5). The oscilloscope is capable of applying a $20\times$ gain (second panel) to the input signal, but unless the signal has a very small amplitude (~ 0.25 V), this amount of amplification will lead to clipping in the sample and hold circuit (see Sec. II-C). The third panel displays the maximum gain the op-amp can tolerate without clipping the signal itself (*i.e.* within ± 12 V). The forth panel shows the maximum gain without clipping from any components in the oscilloscope (*i.e.* within ± 5 V). The fifth panel displays the signal with unity gain (0 dB), the sixth with $0.5\times$ gain (-6 dB) and the sixth with zero gain ($-\infty$ dB). The noise in the zero gain data has a standard deviations of $\sigma_G = 0.1$ V. As this noise is constant amplitude noise, this results in a signal-to-noise ratio (SNR) of ~ 30 dB for a 10 V peak-to-peak signal.

As discussed in Sec. II-B, the oscilloscope is capable of sampling the input signal at four different frequencies. Figure 6 shows the same signal sampled at each of the four different sampling frequencies (150 Hz, 1.2 kHz, 2.4 kHz and 9.6 kHz) compared to the original input signal, shown in the top panel. Note that the sample and hold circuit produces an apparent delay equal to the hold time. This is due to the fact that the output of the sample and hold is exactly equal to the input signal only for the very brief window of time when the circuit is "sampling" the signal (see discussion in Sec. II-C). If the sampling frequency is less than twice the frequency

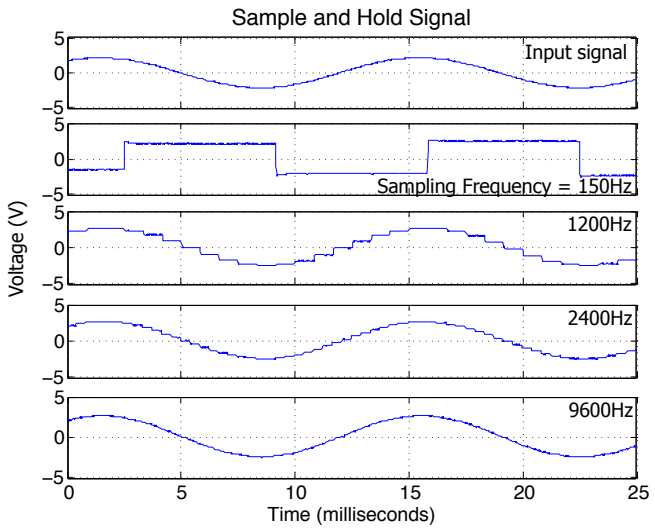


Fig. 6. The output of the sample and hold circuit at various sampling frequencies for the same input signal. The top panel shows input signal, while the remaining panels show sampled signals from lowest to highest sampling frequency.

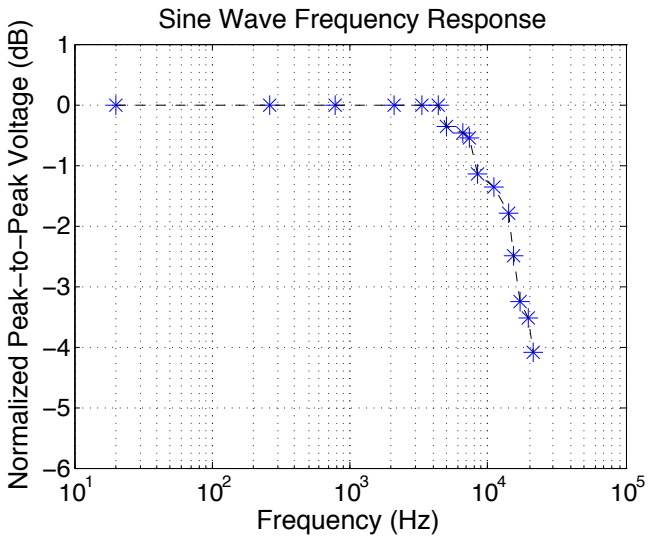


Fig. 7. The oscilloscope's frequency response (peak-to-peak voltage as a function of frequency), measured with a sine wave at frequencies ranging from 20 Hz up to 21 kHz. Sampling frequency is 2.4 kHz. Signal attenuation of -3 dB occurs at $\sim$ 16 kHz.

of the input signal, aliasing will occur. We note that the 150 Hz sampling frequency shown in the second panel of Fig. 6 is close to the Nyquist rate of $\sim$ 144 Hz for the input signal.

To obtain the frequency response of the analog portion of the oscilloscope, we send sine waves of various frequencies into the system and then measure the peak-to-peak voltage of the signal at the output of the sample and hold circuit. This data set was collected at a sampling frequency of 2.4 kHz. Figure 7 shows the peak-to-peak voltage (normalized) as a function of frequency. Our oscilloscope begins attenuating the signal at approximately 5 kHz and the amplitude continues to decay as the signal frequency increases. The "cutoff frequency" of a system occurs when the signal is attenuated by -3 dB, which, for our oscilloscope, is at approximately 16 kHz. This attenuation is due to the capacitor in the sample and hold circuit (see Fig. 16). The amplitude falloff is consistent with that of a low pass filter, as discussed in Sec. II-C.

Fig. 8 shows the output from the sample and hold (top panel) compared with the output of the DACs (bottom two panels). Recall that the signal from the sample and hold circuit is converted to digital and then back to analog. It is this signal which has been reverted to analog that is shown in Fig. 8 in the center panel (green line). The address signal that counts up while the input signal is recorded is also converted to analog (center panel, blue line). Plotting the blue line (T signal) against the green line (Y signal) gives the analog output of our oscilloscope (bottom panel, red line). As discussed in Sec. II-G, the bottom panel is the X-Y mode output. Clearly, this output in good agreement with the sample and hold output signal. Although it is not shown here, Figs. 12 and 13 in the Appendix show the T signal flattening near the start and end of the linear ramp. We are unsure of the source of this flattening, but we have noticed a similar clipping pattern on other devices regardless of power supply voltage. The flattening causes the ends of the sweep of the input signal stored in memory to be "squished" in X-Y mode, since the Y signal is changing while the T signal is essentially constant.

The timing diagram shown in Fig. 11 in the Appendix is the measured version of the simulated timing diagram shown previously in Fig. 2 and discussed in Sec. II-B. The timing signals in this diagram correlate well to the modeled version. One noticeable difference in the noise present in the signal with a standard deviation of approximately 0.06 V or $\sim 1\%$ of the 5 V digital signal. This noise level corresponds to a 35 dB SNR for a 5 V digital signal.

B. Measurements with LabView

Prior to this point, the data shown in this section were captured using the professional oscilloscope and replotted with MATLAB. Here we discuss data captured with LabView via the Arduino which is the end product of our oscilloscope. Capturing data with LabView is a three step process described in Sec. II-E, II-F and II-H. Figure 12 in the Appendix shows the LabView capture with a sampling frequency of 1200 Hz. The bottom plot is

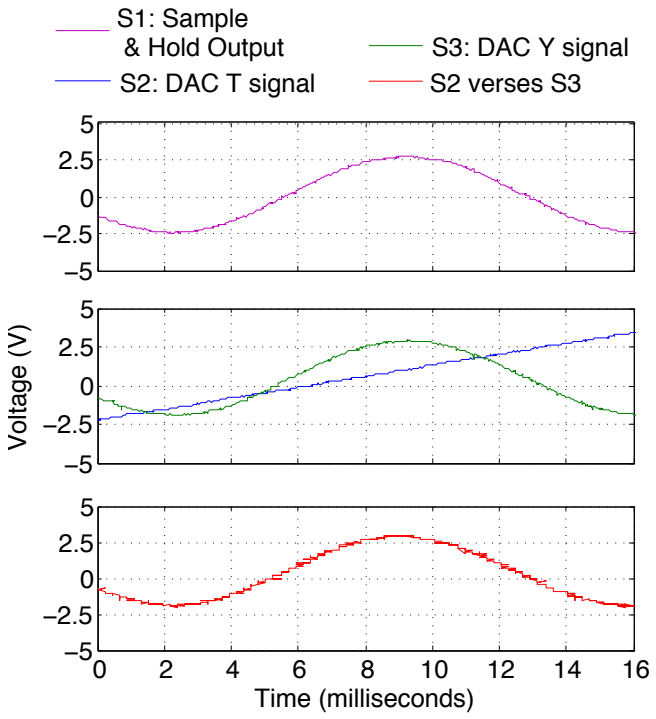


Fig. 8. The outputs of the DACs. The top panel shows the sample and hold output signal. The center panel shows the time output from the DAC (T signal) and the data output from the DAC (Y signal). The bottom panel shows the T and Y signals plotted against each other on the X and Y axes, respectively.

taken from LabView while the top plot was captured with the bench top oscilloscope for comparison. LabView repeatedly plots the same data stored in memory until the user writes new data to the memory. Approximately two sweeps through the memory are shown in Fig. 12. A discontinuity is visible when the signal repeats (this occurs twice in the figure) indicating that the input signal was at different points in its period at the beginning of the sweep and at the end. The top plot shows the T signal counting up (blue line) and the Y signal (green line) being produced by the DACs. Note that time on the X axis in LabView is in milliseconds. The duration of one sweep of the data is approximately 213 ms (256 periods/1200 Hz) for this case. The amplitude of the sampled signal can be at maximum ± 5 V which would correspond to a minimum of 0 counts (-5 V) or a maximum of 255 counts ($+5$ V) on the Y axis (labeled amplitude) of the LabView plot. In our case, the sampled signal has a max/min of roughly of ± 2 V which corresponds to an amplitude between 80 and 180 counts in LabView, which is what we observe. Thus, to use the LabView display as an oscilloscope, one simply reads the X axis as time in milliseconds and the Y axis can be converted to a voltage by subtracting 128 from the amplitude and multiplying the resulting value by the ratio 5 V/128 counts.

Fig. 13, also in the Appendix, shows a second waveform captured in LabView, now with a sampling rate of 9600 Hz. In both of these LabView measurements, the frequency of the input signal was ~ 65 Hz. Notice that the higher sampling rate makes the discretization from the sample and hold circuit less noticeable. However, the consequence of higher temporal resolution is that fewer oscillations are captured to memory for a given input signal frequency, as the size of the memory is fixed. In this case, only slightly more than one period of the input signal is captured and stored to memory. At a sample rate of 9600 Hz, approximately 27 ms (256 periods/9600 Hz) of the signal is captured.

IV. Conclusions and Lessons Learned

In this report we discussed the construction, performance, and operation of our homemade digital sampling oscilloscope. Our final product was a fully operational oscilloscope that allowed the user to adjust the gain and offset of an input signal, sample the signal at one of four possible sampling frequencies, convert the signal to digital and write the data to memory to retrieve and display it in LabView via the Arduino. Our oscilloscope can capture waveforms with a SNR of 30 dB for frequencies up to $\sim$ 16 kHz with ≤ -3 dB amplitude attenuation and up to 4.8 kHz without aliasing. At the highest sampling frequency (9.6 kHz), the memory can hold ~ 27 ms worth of data, while the lowest sampling frequency (150 Hz) allows 1.7 seconds worth of data to be stored in memory. We believe that this sampling frequency flexibility gives this oscilloscope a unique ability to sample waveforms of various frequencies at the required fidelity.

There were several lessons learned looking back at the project. During the oscilloscope construction, we ran into a couple of technical hitches. For instance, the occasional connection was omitted while wiring up analog components. The lesson learned here is that if a component is not operating properly it is most likely due to human error (such as a missing wire or insufficient power), and not a broken component, although the latter might be the instinctive first guess. There is certainly an art to troubleshooting and isolating problems to individual components and eventually finding the problematic (or missing) connection. By the completion of our project we had come to appreciate the importance of color coding our wires to ease troubleshooting and visual comprehension of our circuit board. We also learned that it is essential to have a big picture understanding of how all the components will integrate and work together while still in the planning phase of the circuit. Initially,

we considered each component (*e.g.* the sample and hold circuit) as an individual module, and began building the oscilloscope by completing one module and then moving on to the next. We soon learned that if we did not plan ahead and take into account the interaction between various modules, we would often have to rebuild or rearrange components.

APPENDIX

TABLE IV
INTEGRATED CIRCUIT COMPONENTS

Component	Quantity	Part #
Arduino Nano	1	ATmega328
Op-Amp (package of 2)	3	UA747CN
Bilateral Switch (package of 4)	1	TC4066BP
NAND Gate (package of 4)	1	SN7400N
NOT Gate (package of 6)	1	SN7404N
AND Gate (package of 4)	1	SN74LS08N
Digital Buffer (package of 8)	1	SN74LS244N
D-Type Flip-Flop (package of 4)	2	SN74S175N
Analog to Digital Converter	1	ADC0800PCD
Digital to Analog Converter	2	DAC0800LCN
4-Bit Counter	1	SN74LS163AN
12-Bit Counter	1	CD4040BE
Static RAM	1	HM6116P
Bit Rate Generator	1	MC14411

TABLE V
DISCRETE COMPONENTS

Component	Quantity	Value
Resistor	1	470 Ω
Resistor	4	5.1 kΩ
Resistor	7	10 kΩ
Resistor	1	15 MΩ
Variable Resistor	1	10 kΩ
Potentiometer	1	100 kΩ
Capacitor	2	10 nF
Capacitor	1	68 nF
Capacitor	4	100 nF
Crystal	1	1.8432 MHz
Push Button Switch	1	–
SPDT Switch	2	–

```
const int q9 = 2;
const int PushButton = 3;
const int WriteModeOn = 13;

const int Data1 = 5;
const int Data2 = 6;
const int Data3 = 7;
const int Data4 = 8;
const int Data5 = 9;
const int Data6 = 10;
const int Data7 = 11;
const int Data8 = 12;

int data1Val = 0;
int data2Val = 0;
int data3Val = 0;
int data4Val = 0;
int data5Val = 0;
int data6Val = 0;
int data7Val = 0;
int data8Val = 0;

byte output;

void setup(){
  Serial.begin(9600);
  pinMode(q9, INPUT);
  pinMode(PushButton, INPUT);
  pinMode(WriteModeOn, OUTPUT);
  digitalWrite(WriteModeOn, LOW);

  attachInterrupt(1,WriteMode,RISING);
  attachInterrupt(0,ReadMode,RISING);

  pinMode(Data1, INPUT);
  pinMode(Data2, INPUT);
  pinMode(Data3, INPUT);
  pinMode(Data4, INPUT);
  pinMode(Data5, INPUT);
  pinMode(Data6, INPUT);
  pinMode(Data7, INPUT);
  pinMode(Data8, INPUT);
}
```

```
void loop(){

   data1Val =digitalRead(Data1);
   data2Val =digitalRead(Data2);
   data3Val =digitalRead(Data3);
   data4Val =digitalRead(Data4);
   data5Val =digitalRead(Data5);
   data6Val =digitalRead(Data6);
   data7Val =digitalRead(Data7);
   data8Val =digitalRead(Data8);

  bitWrite(output,0,data1Val);
  bitWrite(output,1,data2Val);
  bitWrite(output,2,data3Val);
  bitWrite(output,3,data4Val);
  bitWrite(output,4,data5Val);
  bitWrite(output,5,data6Val);
  bitWrite(output,6,data7Val);
  bitWrite(output,7,data8Val);
  Serial.write(output);

}

void WriteMode(){
  digitalWrite(WriteModeOn, HIGH);
}

void ReadMode(){
  if (digitalRead(WriteModeOn)  ==HIGH){
    digitalWrite(WriteModeOn, LOW);
  }
}
```

Fig. 9. The Arduino control code.

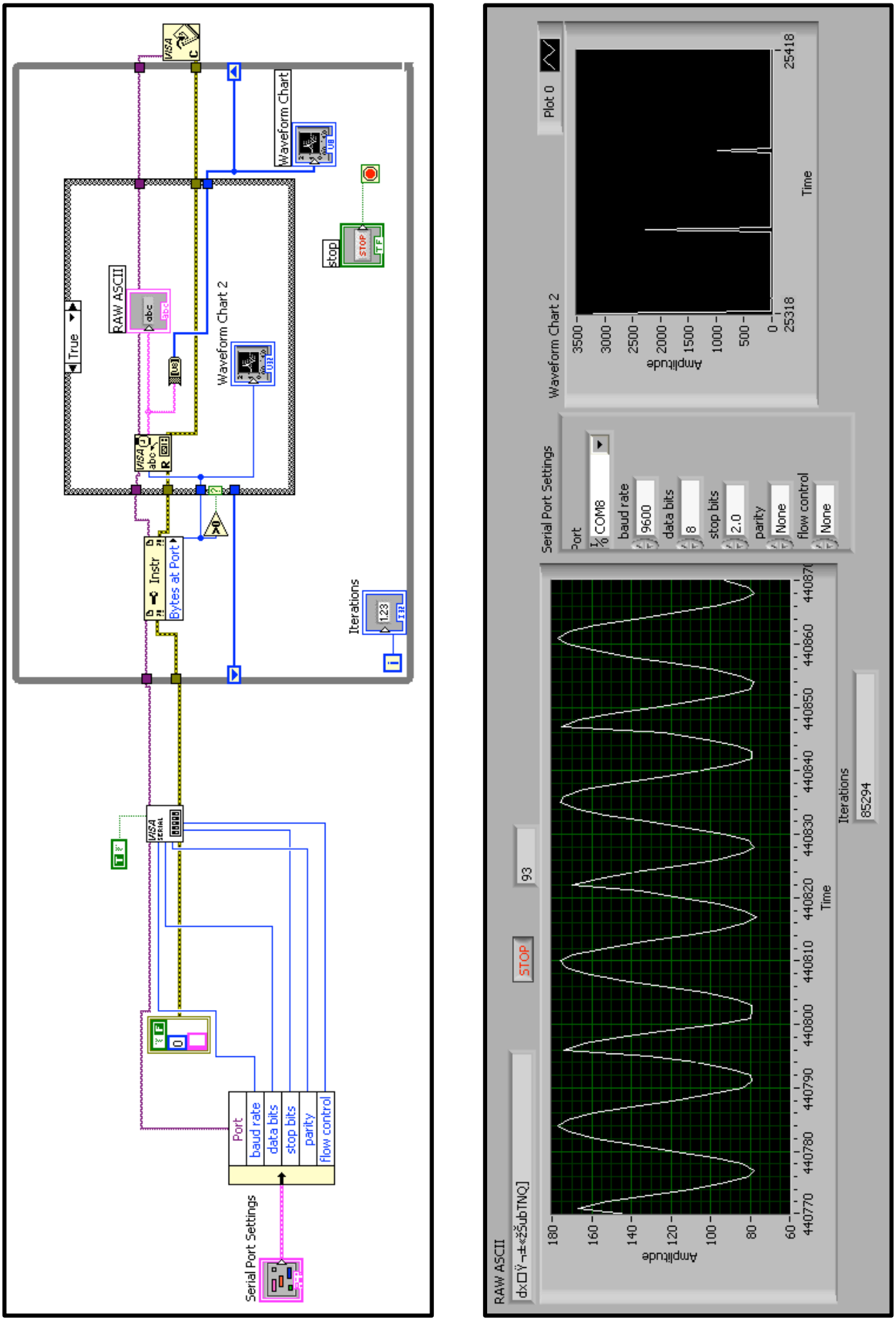


Fig. 10. Top: Block diagram of the LabView program. Bottom: Front Panel of the LabView interface.

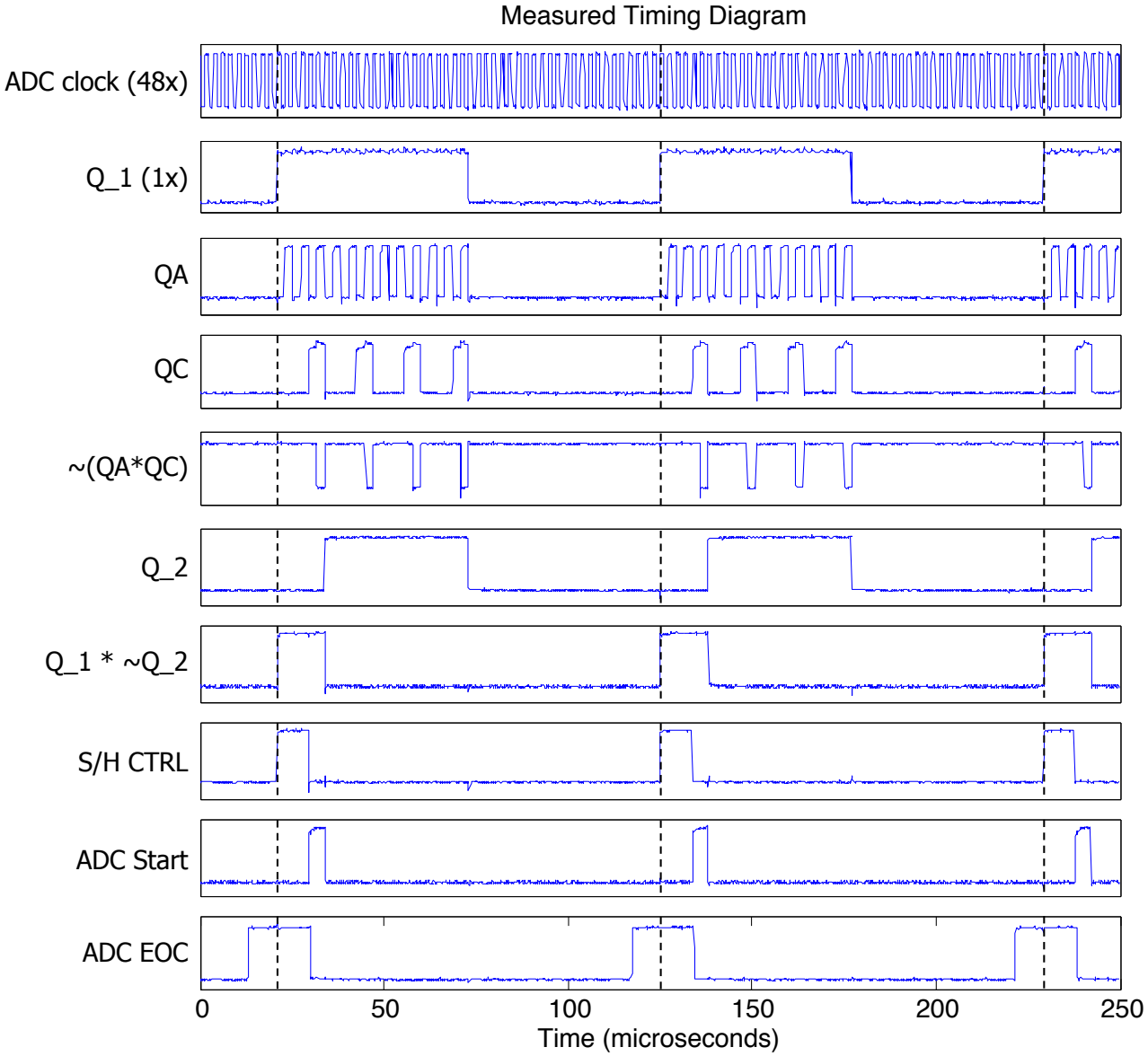
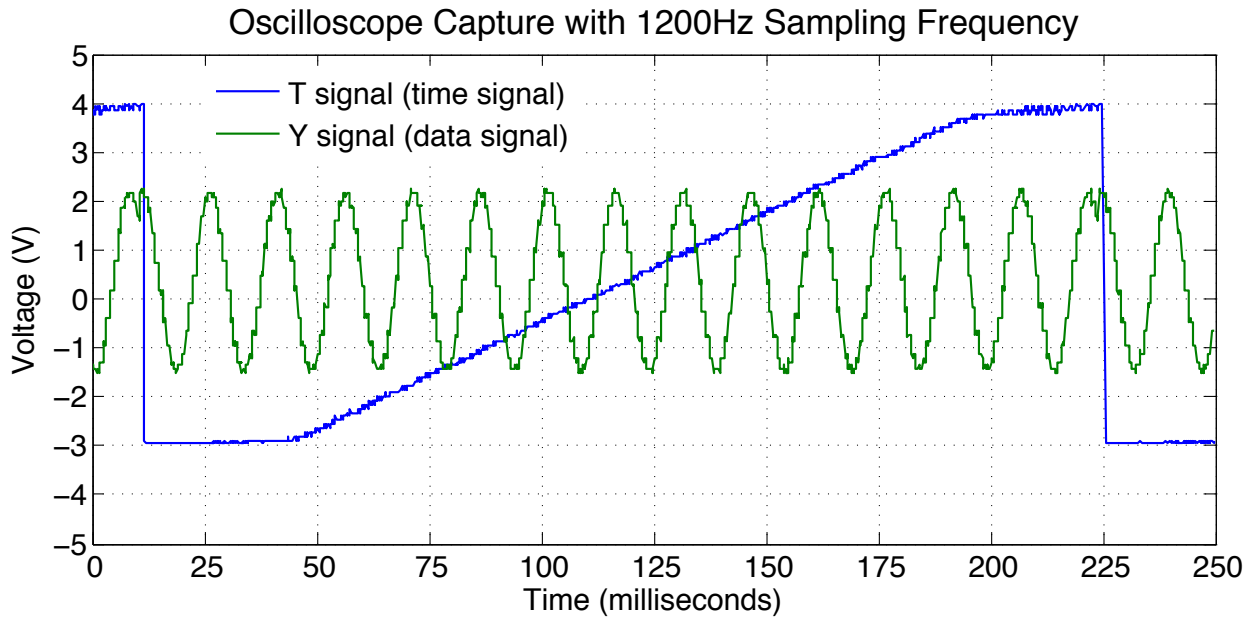


Fig. 11. Measured timing diagram. Shows the measured timing pulses used to control the Sample and Hold circuit and to start the ADC conversion. The timing diagram and the modeled timing signals are discussed in Sec. II-B.



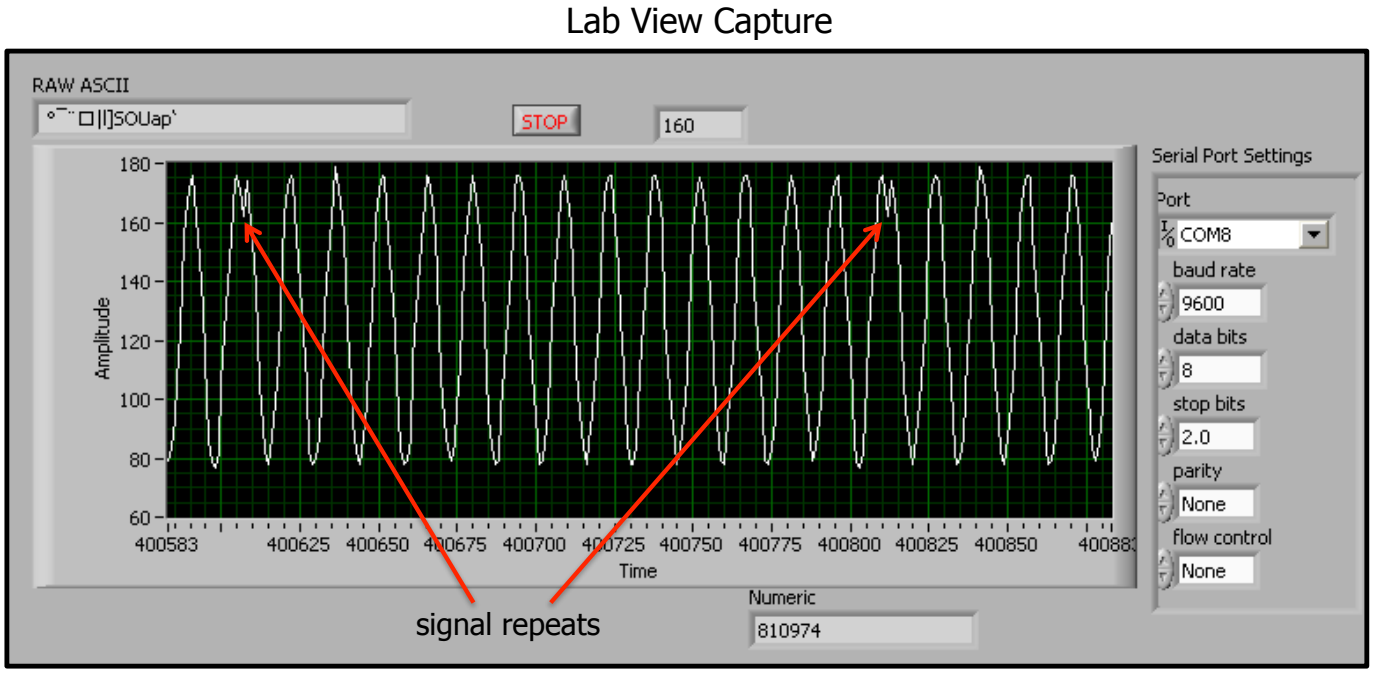


Fig. 12. Oscilloscope data taken with LabView via the Arduino (bottom) compared with data measured after analog conversion via the bench top oscilloscope. Sampling rate: 1200Hz.

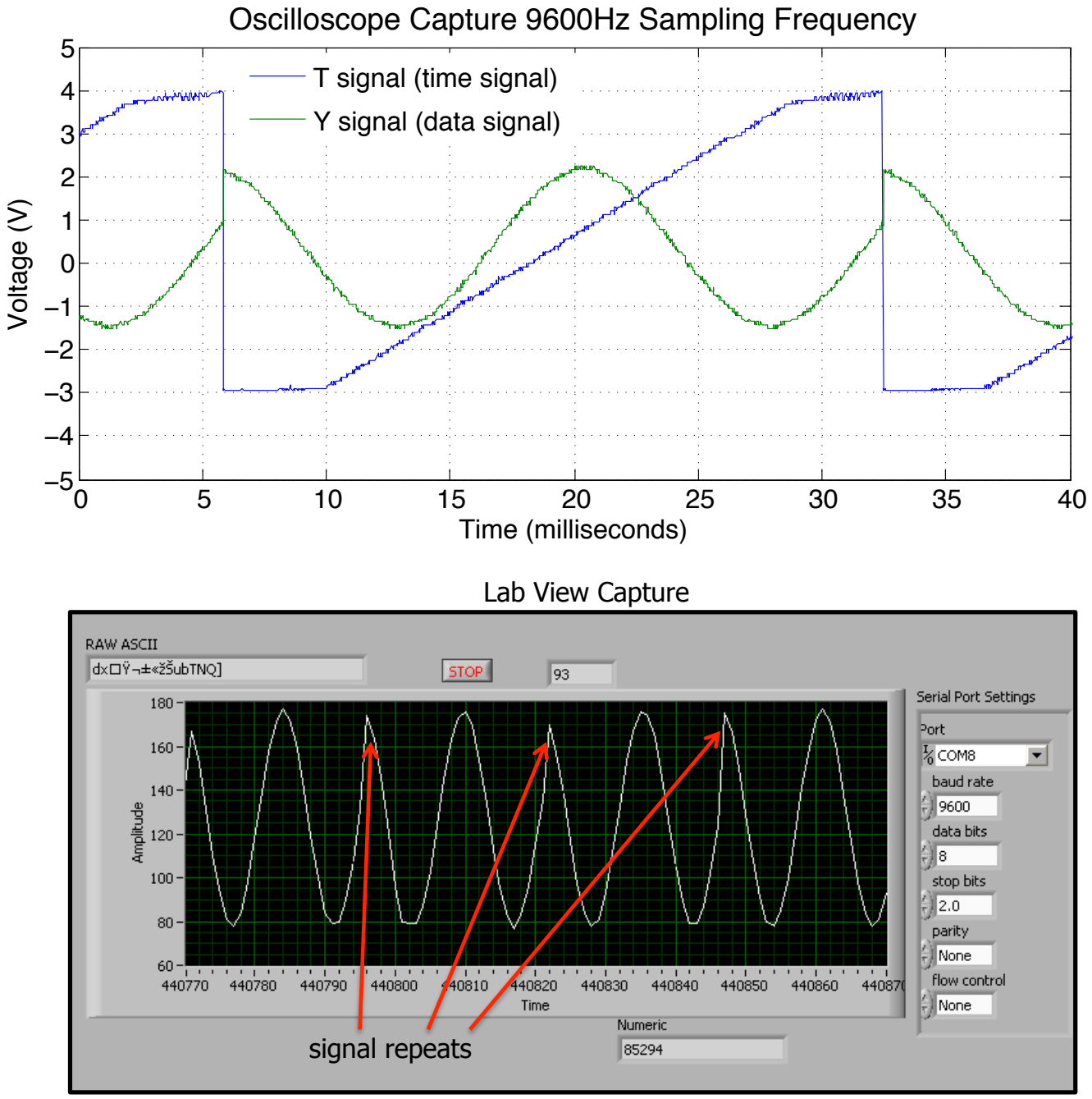


Fig. 13. Oscilloscope data taken with LabView via the Arduino (bottom) compared with data measured after analog conversion via the bench top oscilloscope. Sampling rate: 9600Hz.

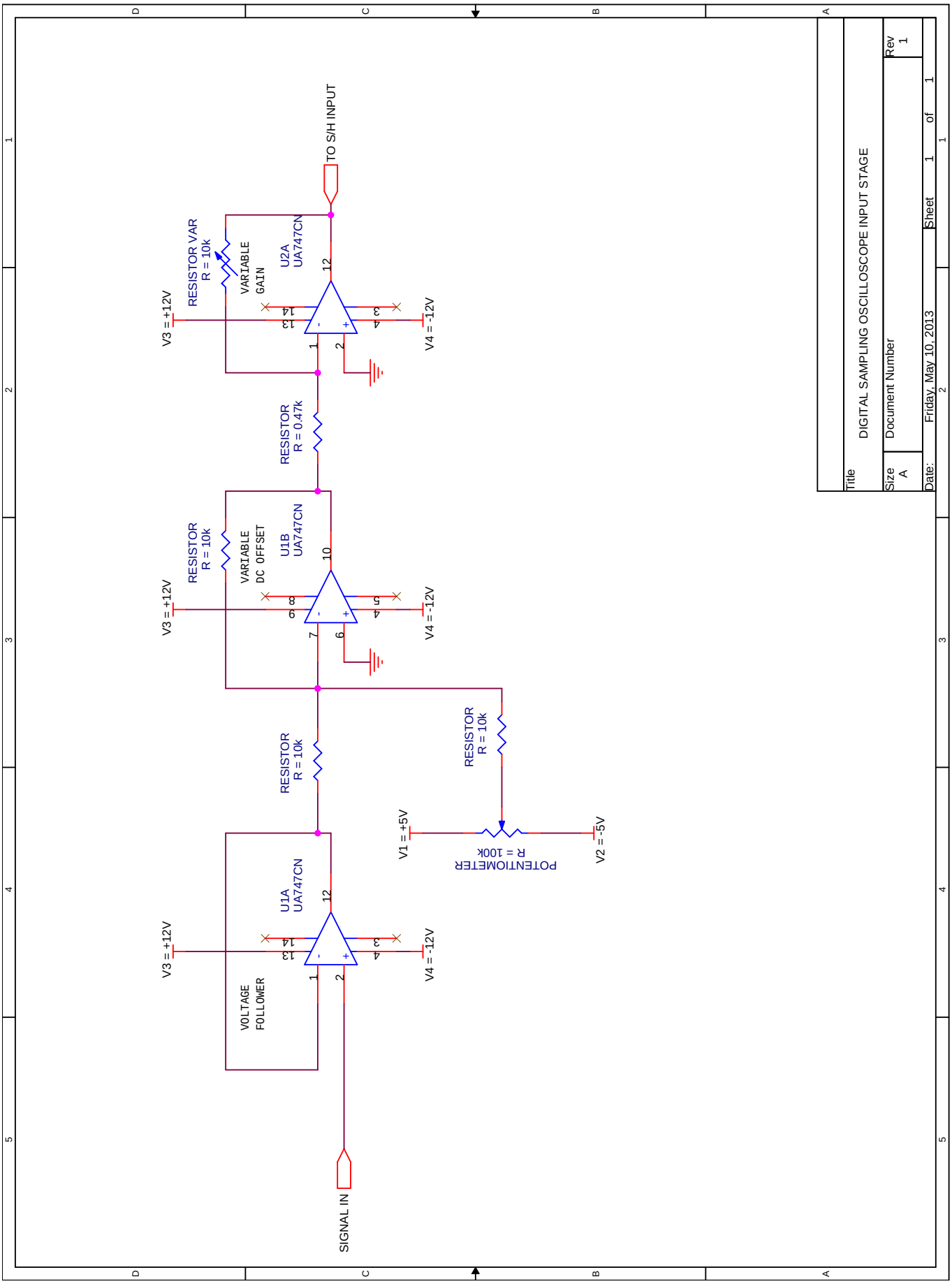


Fig. 14. OrCAD schematic of the Input Stage.

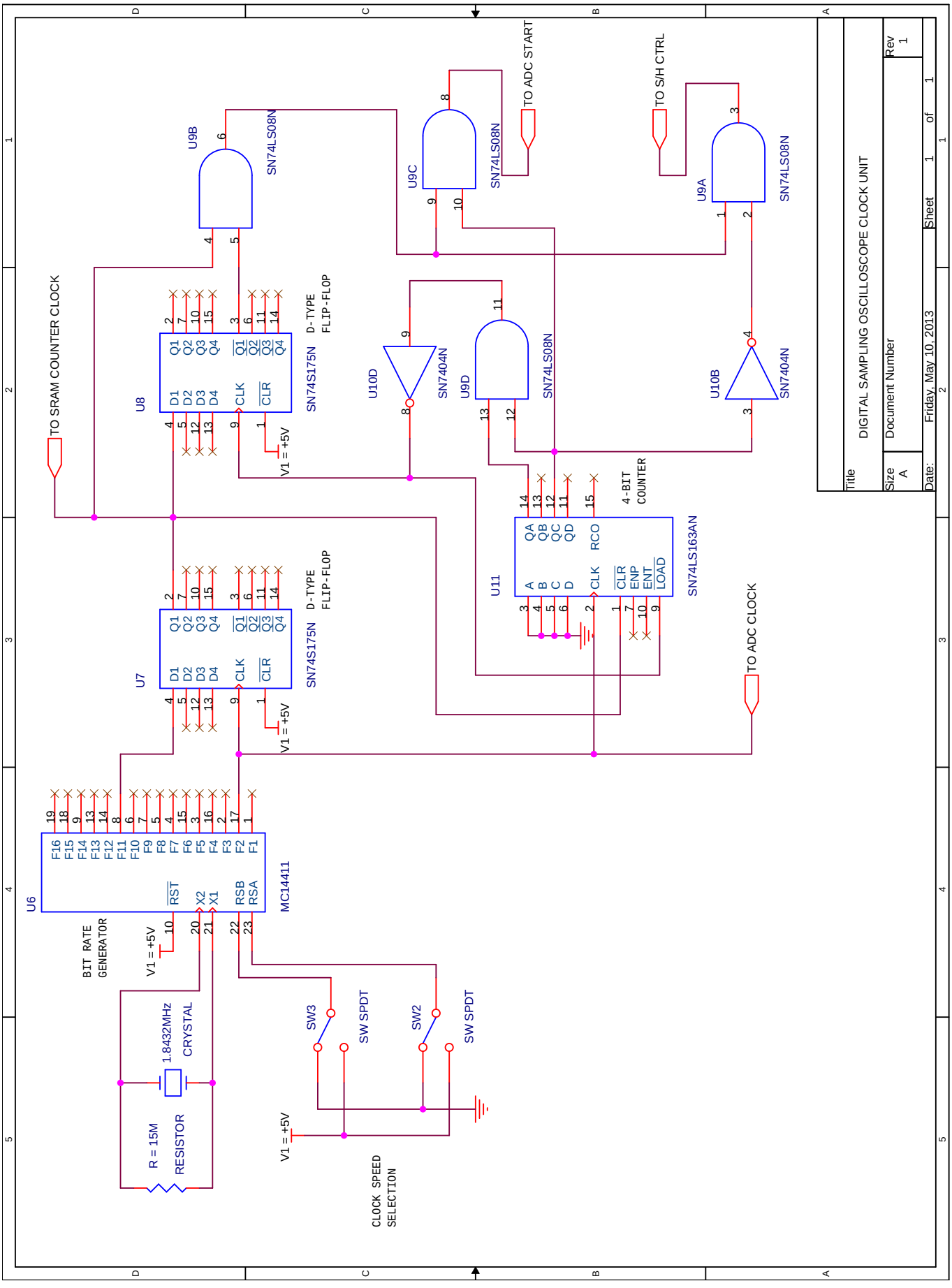


Fig. 15. OrCAD schematic of the Clock Unit.

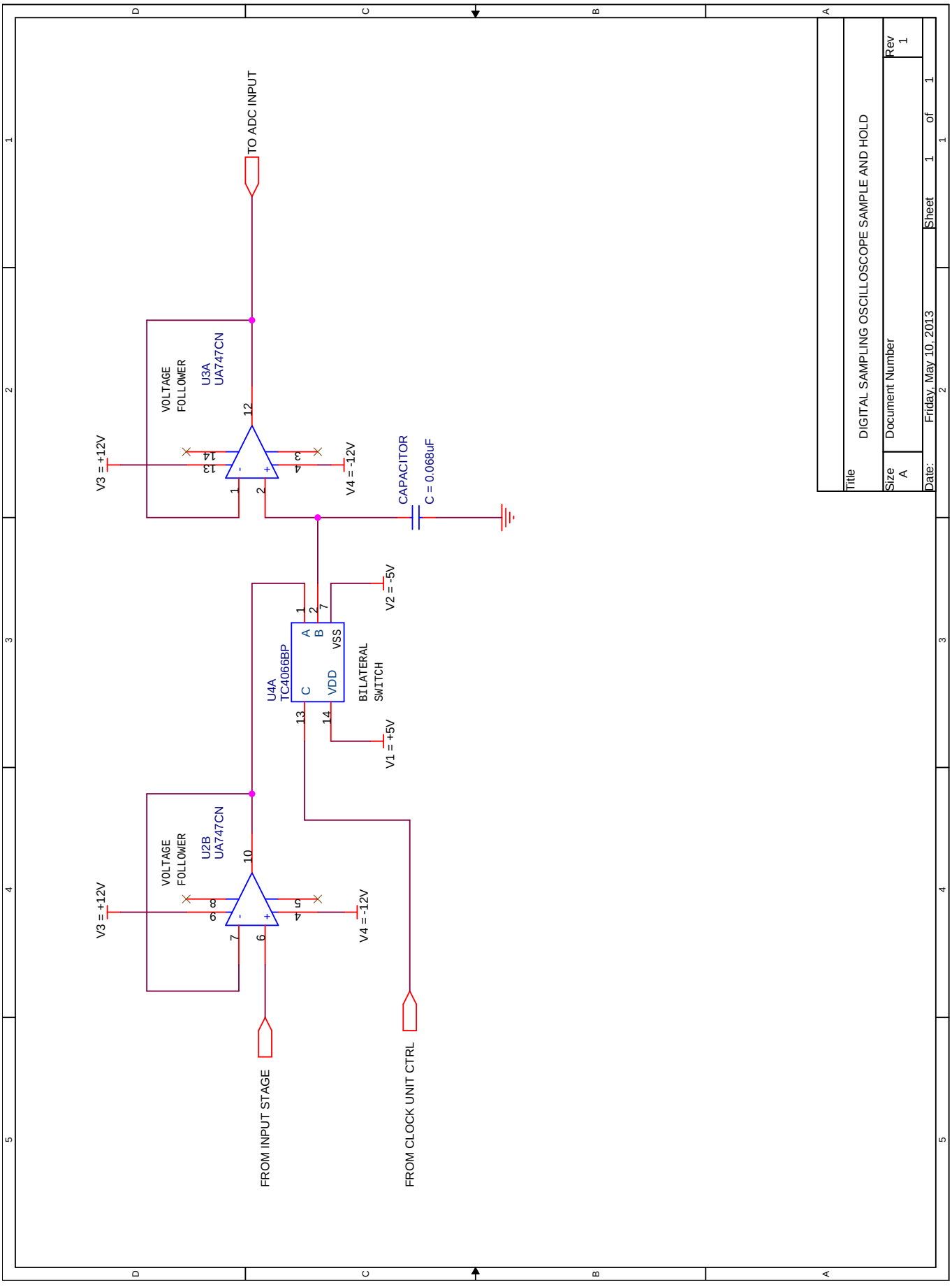


Fig. 16. OrCAD schematic of the Sample and Hold.

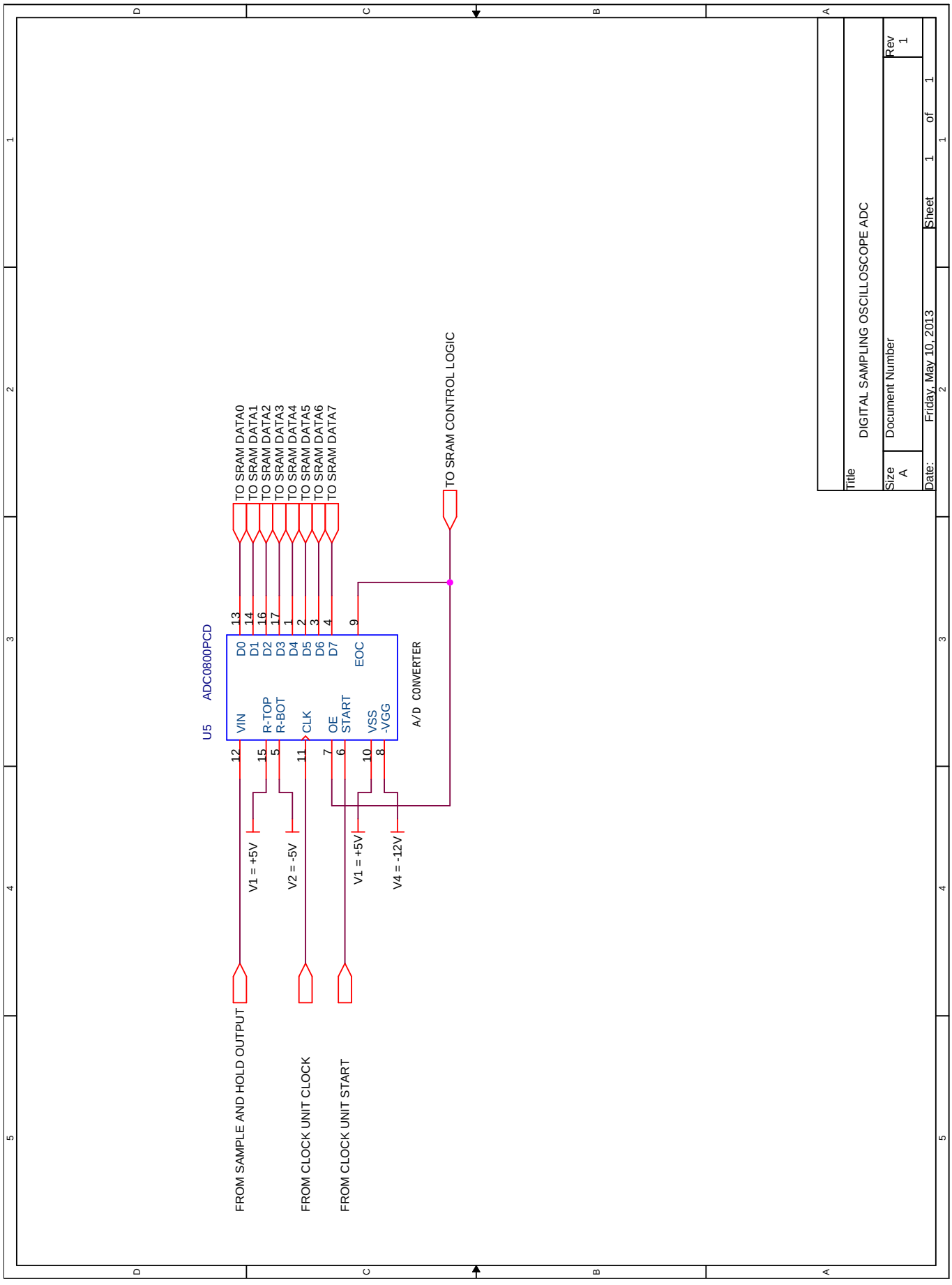


Fig. 17. OrCAD schematic of the ADC.

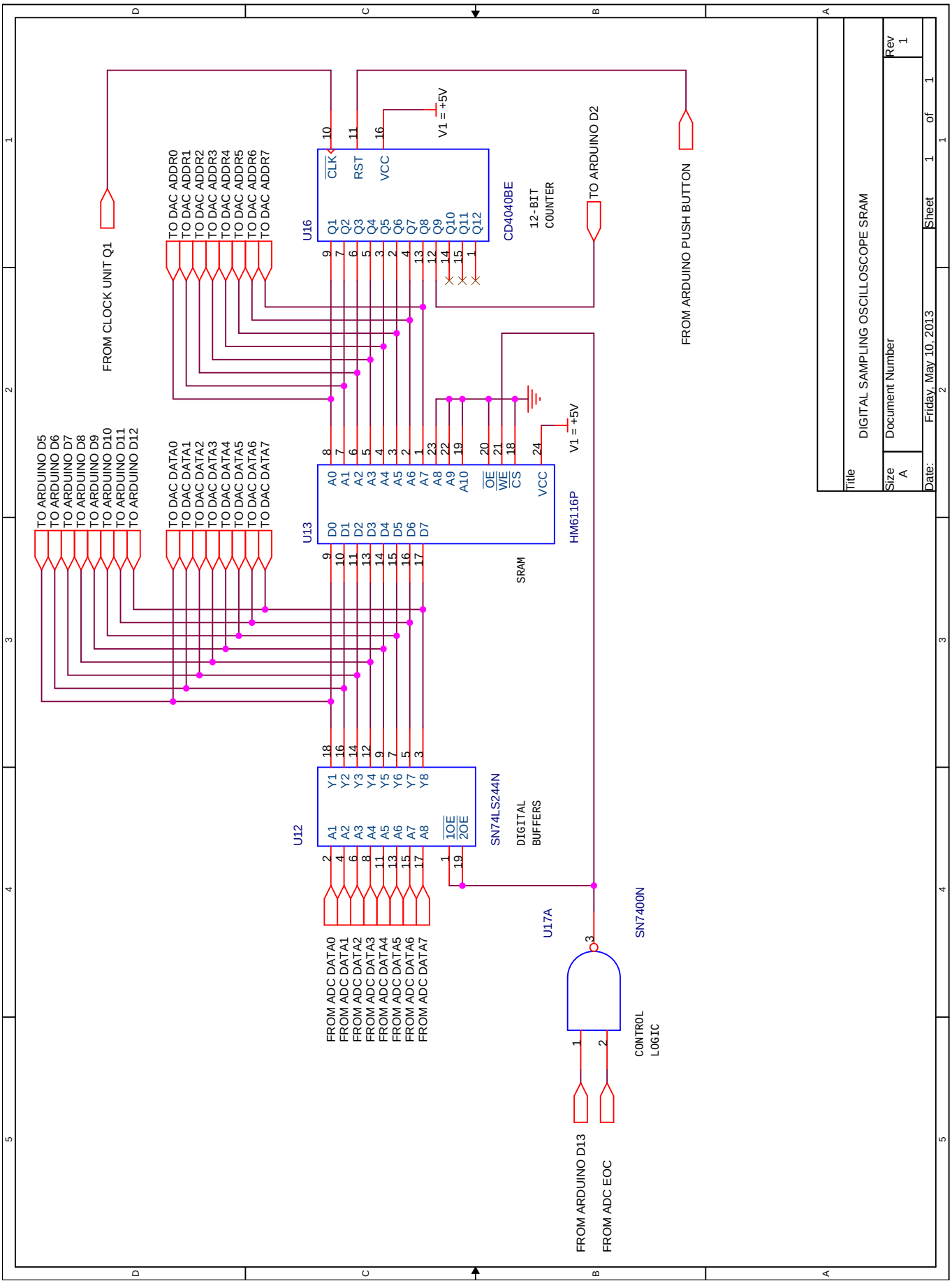


Fig. 18. OrCAD schematic of the SRAM.

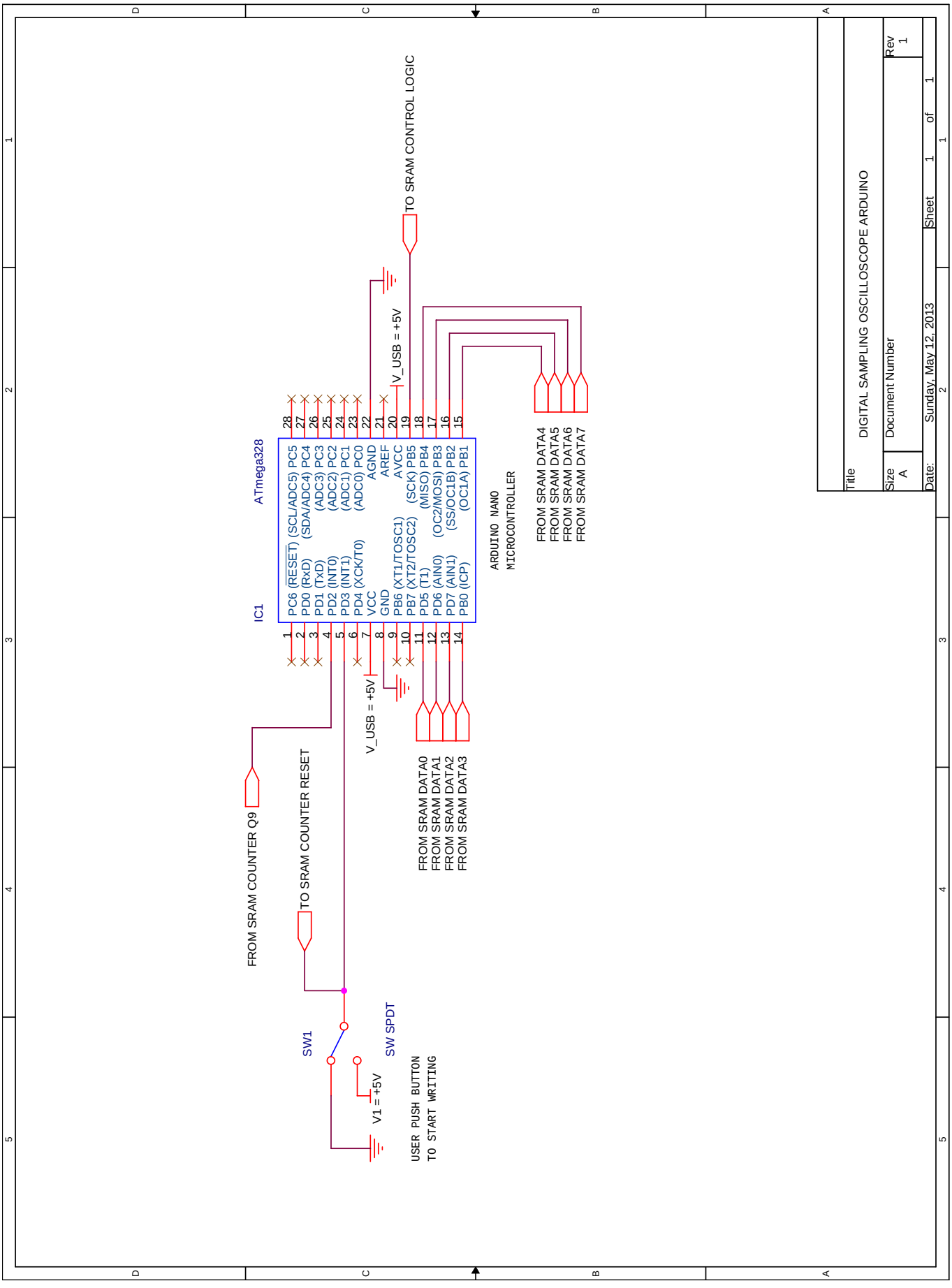


Fig. 19. OrCAD schematic of the Arduino.

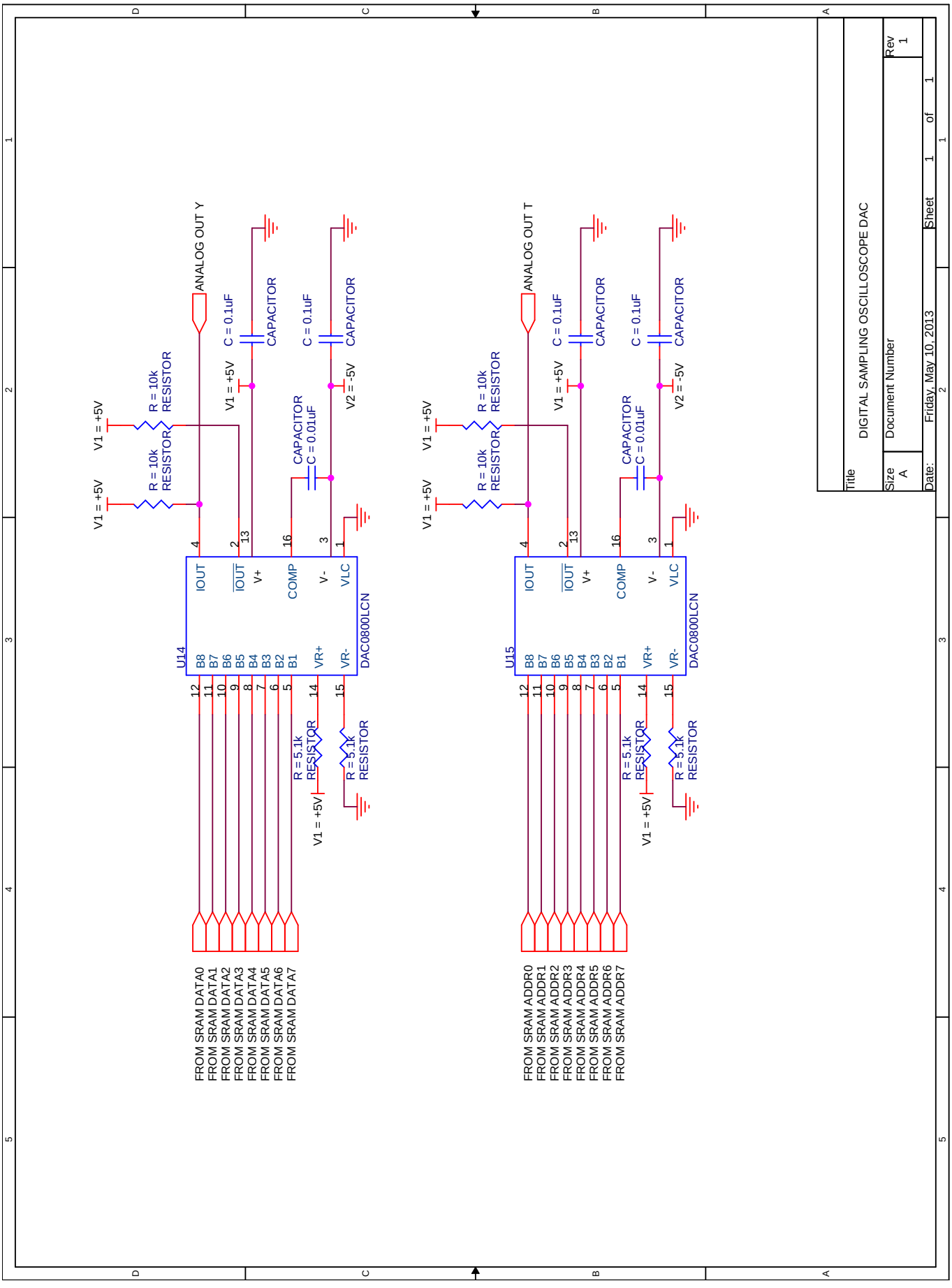


Fig. 20. OrCAD schematic of the DAC.