D2: Report on Integrated Circuit design exercise – team J

Abstract

This is a two-term long design accomplished by the whole team. It began with reading through the specification, collecting message online and discussion in the group. Then the tasks were well distributed among the teammates. The process of the whole design went smoothly, before handing in the final design. Simulation was carried out to make sure the circuit worked properly. All the modules including both hamming encoder and decoder were well arranged on the chip, but there were still relatively large space left on the chip, the manufacturer had to place something on the blank area to avoid sinking in that area. This implied the possibility of improvement in later design. In second semester which is the testing stage, test vectors were designed and all of them passed the checking program in one go. Then they were used to generate SystemVerilog test benches and carry out chip tests at the same time. Simulation in software showed a perfectly functional circuit, however, the real chip gave a different answer. Ring oscillator did not meet the demanded frequency and decoder gave errors from test vector. However the decoder worked when it was downloaded to a CPLD (Il Bagatto), therefore we successfully implemented system test, the digital signals and injected noise waveforms could be read from the oscilloscope easily, and information such as BER (bit error ratio) was available for read directly on the TFT screen controlled by an AVR microcontroller (Il Matto).

Name	e-mail or id	Contribution
Yubo Zhi	yz39g13@soton.ac.uk	Design, test, director
Shuai Shao	ss4g13@soton.ac.uk	Design, chip test
Jiuxi Meng	jm15g13@soton.ac.uk	Design, schematic test
Chris Carville	cc6g11@soton.ac.uk	Design, report proof reading
Diwen Hu	dh1g14@soton.ac.uk	Chip test
Yushuo Liu	yl3c11@soton.ac.uk	Part of design

D2 Project Completion Form For Team J

Team Members:	Credit: Design	Credit: Test	Signatures
Yubo Zhi	35 %	35 %	Yubo Zhi
Jiuxi Meng	20 %	20 %	Jiuxi Meng
SHUAI SHAO.	20 %	15 %	SS
DiWen Hu	5 %	15 %	DiWen Hu
Christopher Conville	15 %	15 %	Chris Conville
Yushuo Liu	5 %	0 %	(We cannot find him before deadline)

Ring Oscillator

	Site (0-15)	Team (A-P)	Design Freq (MHz)	Measured Freq (MHz)	initials
Ours:	9	J	7.7	4.7	BIM
Other:	A	K	8.0	5.6	BIM

Test Vector Results

	Verilog Model Passes Test Vectors	Our Site Chip Passes Same Test Vectors	Other Team (A-P)	Other Site Chip Passes Same Test Vectors	Problems identified with L-Edit Design?	
Inverter	✓	✓	K	✓	—	BIM
4-Bit Adder	✓	✓	K	✓	—	BIM
8-Bit Sequence Recognition	✓ 2159 VEC	✓			—	BIM
Hamming Encoder	✓ 132 VEC	✓	B	✓	—	BIM
Hamming Decoder	✓ 2060 VEC	1056 ERR	N	1756 ERR	—	BIM
Unified Vectors Error Count	✓	1056 ERR				BIM

Our design includes a Hamming encoder: YES/~~NO~~

Our design includes a Hamming decoder: YES/~~NO~~

Hamming encoder/decoder system test result:

	Oscilloscope	Error Correction	BER/BLER	
CPLD Encoder + CPLD Decoder (optional)	—	—	—	
On-chip Encoder + CPLD Decoder	✓	✓	✓	BIM
On-chip Encoder + On-chip Decoder	—	—	—	

Details of any identified problems with L-Edit Design:

(where problems are identified additional deliverables will be required)

Hand-in

Files and report due by 16:00 on Friday 8th May

Design Reviewed by

Signature I. M. Neil Date 1/5/2015

Introduction

This design exercise concerns the design and simulation of individual circuits and modules for the purpose of integration on a chip during fabrication. A total number of 5 digital circuits were designed by our group, these include an inverter, ring oscillator, 4-bit adder, sequence recogniser, hamming encoder and hamming decoder.

The inverter is a simple CMOS inverter, the simplest element that must work, used to aid automated chip check.

The ring oscillator is an oscillator formed by inverter chains, our target frequency is 7.7MHz. It was designed by Chris Carville (cc6g11) and Jiuxi Meng (jm15g13).

The 4-bit adder is a general adder with carry in and carry out, designed by Shuai Shao (ss4g13) and Chris Carville (cc6g11).

The sequence recogniser accepts a stream of bits as input, and signalling when it found the predefined sequence, designed by Yubo Zhi (yz39g13) and Jiuxi Meng (jm15g13).

The Hamming encoder together with hamming decoder utilise the Hamming error correcting code, to reduce data error during transmission, by correcting simple data errors. Schematic designed by Yubo Zhi (yz39g13), chip layout designed by Yubo Zhi (yz39g13), Shuai Shao (ss4g13) and Jiuxi Meng (jm15g13).

Full chip integration by Yubo Zhi (yz39g13).

Ring oscillator test vector was designed by Shuai Shao (ss4g13), all other test vectors were generated by programs written by Yubo Zhi (yz39g13). Jiuxi Meng (jm15g13) with Chris Carville (cc6g11) did schematic test using test vectors, while Shuai Shao (ss4g13) with Diwen Hu (dh1g14) did chip test. System test and test program on AVR microcontroller by Yubo Zhi (yz39g13).

Diwen Hu (dh1g14) was not actively involved in the design process, mainly because of he was a direct entrance second year student just enrolled in the university at that time, did not have the knowledge prepared for his first laboratory session yet. But he inspected the whole design process, learning new things.

Yushuo Liu (yl3c11) was not actually involved in the design process and failed to attend the 2 testing laboratory sessions. Therefore we decided to give him 5% for design and 0% for testing. Chris Carville was also absent in the second testing session due to other commitments which could not be re-arranged but gave notice for his absence in advance.

This report was written by Yubo Zhi (yz39g13), Shuai Shao (ss4g13), Jiuxi Meng (jm15g13) and Chris Carville (cc6g11).

Design

Inverter

The inverter in this design is used to aid automated checking and feedback in later chip test. If the inverter is not working, then obviously the chip will not work either. It is a basic test to check for colossal failures in design and particularly layout, for example crossing of rails in the layout design or shorting of supply and ground rails.

Ring Oscillator

A ring oscillator was designed by using an odd number of inverters to form a chain. The output of the last inverter is the logic NOT of the first input and it is fed back to the first inverter to give a continually switching output. The circuit oscillates at a certain frequency due to the propagation delay of each CMOS inverter. Since the oscillation frequency is dependent on the number of stages and the delay time of each stage, different inverters (e.g. NAND gate, NOR gate) were used to meet the specification. The output of the gates is fed into row of flip-flops to step down the frequency by dividing by 2 in each case to get the oscillation to the required range, the final changes/tweaks were made by choice of gate type in the initial oscillator stage. The final design of the ring oscillator is shown in 00.

- Base Frequency = 5 MHz
- Offset Frequency = Team ID × 300kHz = 9 × 300kHz = 2.7MHz
- Design Frequency = Base Frequency + Offset Frequency = 7.7MHz

4-bit Adder

A 4-bit adder was designed by using 4 full adders. It requires two 4-bit input and can generate one 4-bit output. Unlike other full adders, the full adder in this design did not use any AND gate for the calculation of carry out bit. AND gate cannot be found from the layout components, which may be caused by hard fabrication and high delay. Instead of using three inverter for one full adder, all AND gates in the full adder design were replaced by NAND gates. This change would not affect the result and save more space for other circuits.

Sequence recognition

Target sequence: 11100101

The aim of the sequence recogniser is to assert whether a certain incoming sequence from a large stream of data matches a required pattern. It is able to detect the overlapping from the stream as well. So, if for example it is to detect the sequence 110011 the sequence 11001110011 would provide 2 matches.

The design of the sequence recogniser begins with an ASM chart, as shown in Appendix B. As a bit stream is input to the state machine, the states progress linearly as matches are made with the target sequence. Overlapped detection was done in the state machine explicitly by not returning to check the first bit after a mismatch occurred or after a full sequence matched, but return to a position where some previous input bits might match the beginning of the target sequence. For example, the last bit of our target sequence was 1, which was also the first bit in the sequence, so after a sequence matching, the state machine should return to checking the 2nd bit, but not restart from 1st bit. Even after a false bit, for example if the sequencer detected an input of *1110011*, there would be no match, but the state machine does not return to checking the 1st bit, since the last 2 bits input match the first 2 of the target sequence. This covers the machine from missing a target sequence as it is actively asserting input patterns rather than checking one pattern at a time before resetting.

Hamming Encoder

Hamming encoder takes 4-bit parallel input data and convert them to an 8-bit serial output data. In between the original data there were the parity bits inserted for detecting and correcting errors in decoder.

This design contains a state machine and an ALU. The state machine was a simple 3-bit counter that counts from 0 to 7 after start signal.

The encoding process could be described as, the counter in the state machine controls which bit to be output, each number from the counter corresponding to a different combinations of channels in bit select multiplexers.

By listing the truth table of parity bits, the combinational logic of the input and output can be derived using K-map, summarised as Hamming encoding matrix (modulo-2 matrix product):

$$\begin{pmatrix} \overline{b_0} \\ b_1 \\ \overline{b_2} \\ b_3 \\ \overline{b_4} \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{pmatrix}$$

It could be seen from the algorithm that bit 1, 3, 5, 7 in output sequence were came from input data directly, while others were came from XOR results of 3 data bits. Therefore the least significant bit from state machine counter output was used to switch between direct data bits or XOR results.

The switching between direct data bits to output bit stream was done by using 2 level cascaded multiplexers, switching between 4 data bits.

By further investigation of parity bits XOR calculations, the calculations could be rearranged as:

bit	XOR input 1	XOR input 2	XOR input 3
b_0	$\overline{D_0}$	D_2	D_3
b_2	$\overline{D_0}$	D_1	D_3
b_4	$\overline{D_0}$	D_2	D_1
b_6	D_1	D_2	D_3

Therefore, only 2 2-bit input XORs were required in the parity calculation to form a 3-bit input XOR, with each input selected from only 2 data bits, controlled by state machine higher 2-bit counter output, using multiplexers and NAND gates. This was great because the layout area occupied by XOR gate was nearly 2 times than other gates, this design uses minimum number of XOR gates.

Appendix E shows the encoder schematic designed by using the method described above.

Hamming Decoder

The Hamming decoder reads 8-bit code words from serial interface, then decodes this message back to 4-bit data. Parity bits in 8-bit code words were used to identify and correct possible bit errors during transmission though parity check matrix and the calculated syndrome. It is always able to recover data from 1-bit error in code words, can only identify 2 errors but is unable to correct them and may report incorrect data as correct for more than 3-bit errors.

Decoder syndrome algorithm (modulo-2 matrix product):

$$\begin{pmatrix} s_A \\ s_B \\ s_C \\ s_D \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix}$$

Syndrome error correction scheme as follows:

Syndrome / $s_A s_B s_C s_D$	Inference
1111	No error.
0000	Error in $b_1(D_0)$
1000	Error in $b_3(D_1)$
0100	Error in $b_5(D_2)$
0010	Error in $b_7(D_3)$
xxx0	Error in parity bits
xxx1	Multiple errors, cannot detect

Syndrome D was used to determine whether received data is valid or need correction, then flip the corresponding bit by using XOR gates, with inverter and NOR gates for determine bit number. The data were decoded by using combinational logic.

After data has been decoded, the validity signal of data and the data itself were kept until next data received by using serval D-type flip-flops. The state machine inside the decoder is a simple counter that counts received data bits though serial data input, then determines when to update output buffer to data decoded.

For reducing delay between the last data bit received and when data is ready, the shift register to handle serial data input stores only 7 bits, with the 8th bit directly sourced from input data line. Since output data would be held though flip-flops and only update at clock rising edge, changing in 8th data during clock cycle would not affect current output, so this design works without problems.

Layout Design

Our group layout design was finished part by part. The size of every single circuit was determined by the estimation of the whole chip design. The process of layout design is achieved by using L-Edit and referencing the schematic of each circuit. Beside the design rule checking of the software, there are several other considerations that need to be made. Most of the wire that connect each component will play the role of transmitting signal. Those wires should be designed as short as possible because of the propagation delay. For instance, the Hamming decoder is depended on the clock signal. A little propagation delay will manifest in the incorrect result. In addition, components that need to be connected to the power V_{cc} and Ground should be placed near to the edge of the chip. It will give convenience of drawing power and ground wires.

After each part of layout has been finished, it should be extracted as a SPICE file and be simulated by OrCAD with schematic. If the simulation shows correct behaviour according to the signal that given, the layout design will pass the check and the test should translate well to a fabricated chip. The whole chip integration combined all parts of layout that simulated well, which is shown in Appendix B. There is one problem with whole chip layout of our group. The available space on the chip was not completely filled, which could result in a defect after fabrication (polishing the blank 'softer' areas

could cause damage). In order to avoid that, some supporting structures were added to the empty region by the manufacturer.

Testing

Test vector generation

The test vectors must be correct, a failure to make correct test vectors will result in finding errors in a circuit which is otherwise operating correctly. They can implement a functional test of the circuit to ensure the specification is met or they can be exhaustive to ensure the system can reach all available states. Testing of the physical design and simulated design must also be cross compared. If a similar error is found in each the design can be assumed incorrect, otherwise manufacturing defects become more likely.

All the test vectors were generated in the right format using C++ program.

Circuit	Number of vectors	Circuit	Number of vectors
4-bit adder	512	Hamming encoder	132
Sequence recogniser	2159	Hamming decoder	2060
Combined	4218		

The test vector for adder, encoder and decoder was generated by listing all the possible combinations of the input and checking if the output is correct or not. This can be done easily by writing several cascaded loops in the program. They are indeed an exhaustive test but at the same time the most efficient way of testing since they are not very complex circuit.

For sequence recogniser, in order to test whether the circuit can handle overlapping in the incoming sequence, the test vector was generated by combining two parts together. There are two loops in the program, first one is used to output and print the first n bits of the sequence, the second loop which is also the sub-loop of the first one is used to output and print the full sequence right after the first n bits. Every single test is then different in length and should have the form of

First n bits of the sequence + original sequence

They are vectors performing a function test because the sequencer does not care the size of the input, which means an exhaustive test can have infinite number of vectors.

Inverter

Working perfectly.

Ring Oscillator

Oscilloscope capture in 0, Figure 1.Targeted frequency was 7.7MHz, the frequency in schematic design simulation was 7.73MHz, but the actual frequency on chip was 4.61MHz. This discrepancy is due to parasitic capacitance and inductance in the circuit. These manifest due to the close wiring in the device which are effectively varying charges at a distance to one another, IE a capacitor. The sharp 90 degree turns in the layout wiring (and also vias between layers) can contribute to an inductance. The discrepancy is more likely contributed to the fact that individual devices on the chip are not related to the simulated versions, where different propagation delay times are likely. Compound these discrepancies with many devices on the chip and the oscillation frequency will differ largely.4-bit Adder

The adder on chip works without any problem. Tested though test vectors for all adder calculations.

Sequence Recognition

Sequence recogniser works without any problem. Schematic and on-chip tested by test vectors.

Hamming Encoder

Hamming encoder works without any problem. Schematic and on-chip tested by test vectors.

Hamming Decoder

The Verilog file extracted from Hamming decoder schematic design gave no error from the test vectors, unfortunately the on-chip decoder was not fully working, failed part of the same set of test vectors. There were no errors with ready, validity, error signals, but some decoded data were wrong.

The schematic had been checked matching the layout design, suggests it was a design problem. The method of keeping decoder output was a few flip-flops, however their clock was sourced from a signal from state machine instead of the global clock used by other flip-flops, because of trying to save some layout space by not using multiplexer. However the design became not fully synchronise, the delay between clock update and combinational logic probably was the cause of the problem.

Chip Test

The Southampton "superchips" were fabricated and packaged up with all group designs in a single device. Every group need to switch the site number in order to test their own integrated circuit. The site number that arranged for our group J is 9. This test board need a standard power supply voltage of 5V. Considering about the safety, the current limit of power supply unit was set to 100mA. The process of chip test was achieved by the software, IC Design Tester Software User Interface, given by the university. After the test board was well connected, test vectors should be loaded and run tests. If the result shows no error, the circuit will be regarded as a successful design.

Via test vectors designed by our group, the circuit of inverter, 4-bits adder, 8-bit sequential recogniser, and Hamming encoder are tested successfully. In order to verify the suitability of our test vectors, we tested all other group's adder and Hamming encoder circuits via our test vectors. After discussing with them, we found that the result shown by our test vector was correct and suitable to determine any errors in design or manufacturing.

System Test

System testing was achieved by utilising a microcontroller to generate a sequence of continuous data into an encoder, then decoded by a decoder, to check if it can decode the data correctly. Some random noise was also added to the serial data transmission by using a XOR gate with generation program on microcontroller. The TFT display library used in microcontroller code was available online at [1].

Since the encoder on our chip was working, but decoder was not, we were using on-chip encoder with a CPLD decoder, by loading the working Verilog file extracted from schematic onto the CPLD.

Screen capture of such connection and data sequence were shown by Figure 2, in Appendix B. The figure shows a sequence of 0x0 to 0xF fed into the encoder, which then generated a sequence of bits by hamming encoding algorithm, had some noise injected, finally decoded through the decoder, results checked by microcontroller.

Before injecting errors to the data line, the decoder could decode any data from the encoder without any errors. Also the sequence of data output from encoder were checked by comparing to the table of desired code word, they were all correct. These suggests our on-chip encoder and CPLD decoder works.

After adding random error generation and injection, by counting errors on microcontroller, the results were, for a bit error generation rate of 1.28%, invalid data rate were 0.98%.

The results show the effectiveness of hamming coding error correction magnesium. For 4-bit data transmission, a bit error rate of 1.28% means the data would be corrupted at a chance of 5.12%, but the hamming coding algorithm effectively reduced that to 0.98%, approximately 5 times lower than direct transmission.

Conclusion

The design exercise was successfully completed with the use of team planning, design, testing and communications. The exercise highlights the role of modelling during the initial design stage and the testing of a third parties circuits, as they are more likely to test the device thoroughly and eliminate the confirmation bias associated with testing one's design. Successful modelling is key to ensuring a reference point can be compared against to correct errors or find manufacturing errors that could be due to the fabrication process and ultimately rule out bad design. This has been highlighted particularly during the testing phase.

Apart from decoder, our other circuit design parts work perfectly. The problem with decoder is the design was not fully synchronised, some of the flip-flops were not clocked from the global clock, which could cause all sort of strange problems. This could be solved by using global clock for output buffer flip-flops, with the value updated by using a multiplexer for selecting flip-flop inputs.

The parts in chip layout could be placed more tidily together. Making more use of the space available on the chip would prevent manufacturing defects and help separate each individual circuit, saving manufacture cost as well. Care must be taken for example in decoder and encoder designs to minimise crosstalk between very narrowly spaced wiring.

We successfully adapted test vector mechanisms to test schematic design and chip test. These are not designed to be exhaustive. From an economical view simplified vectors are beneficial in scale manufacturing and testing, although not the case for this exercise, efficiency is a major part of testing. More exhaustive tests have been seen with tens of thousands of vectors, but the testing time for this is minutes which will greatly increase production costs. It may have been possible to find more errors in design with these exhaustive tests but we are confident our functional tests have adequate vectors, this is based on our testing of other team chips with them. The most exhaustive tests can find errors which may mean functional vector tests are not suitable and need to be revised, or they may find rare errors which do not effect performance. Implementing scan paths can also greatly improve testing, but are a design process in their own right.

System test combining encoder and decoder was also successful.

References

[1] Z. Yubo, “GitHub: Il-Matto,” [Online]. Available: https://github.com/zhiyb/Il-Matto.

[2] U. o. Southampton, “D2: Integrated Circuit Design Exercise,” [Online]. Available: https://secure.ecs.soton.ac.uk/notes/elec2205/D2.

Appendix A. Oscilloscope captures

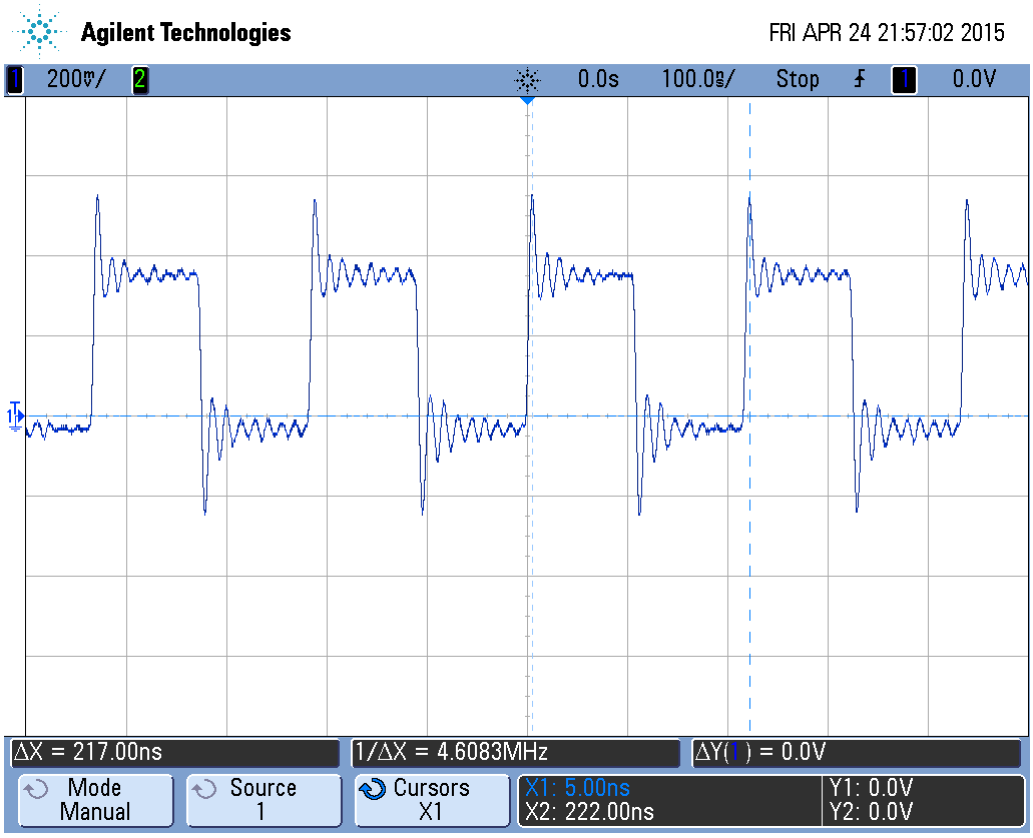


Figure 1 Oscilloscope capture of ring oscillator chip test, frequency shown on bottom cursor measurement was 4.61MHz

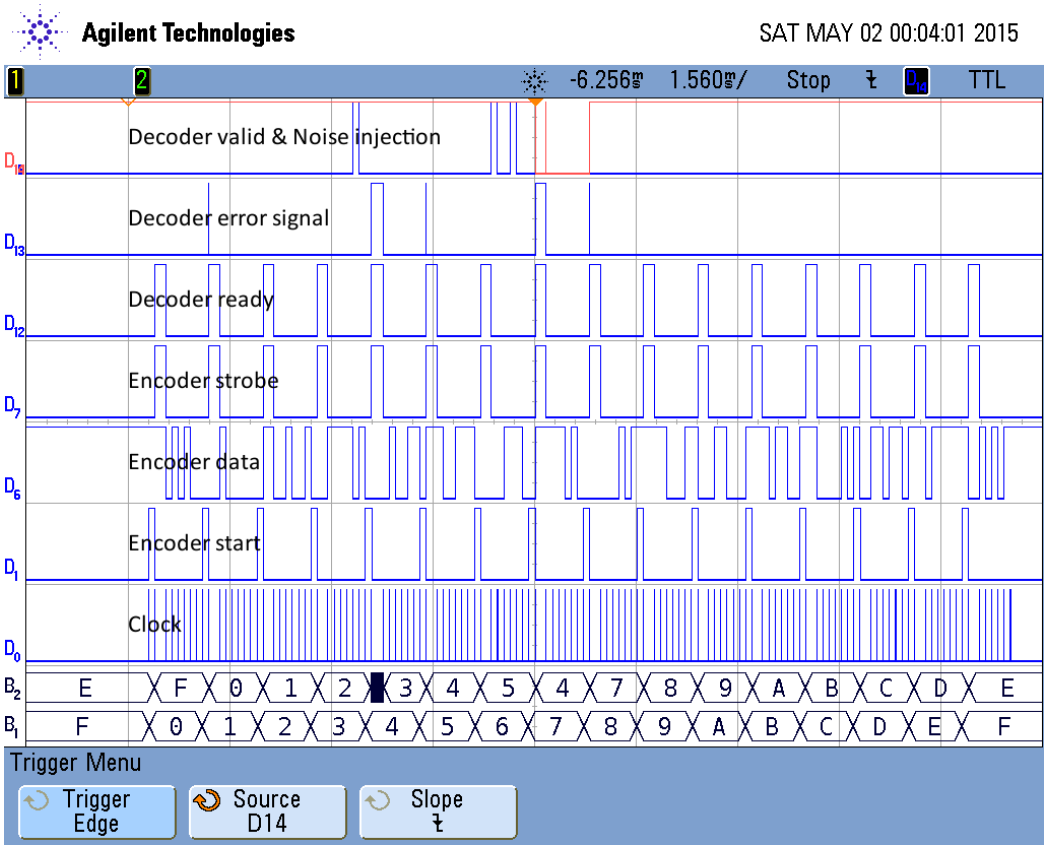


Figure 2 Oscilloscope capture of encoder & decoder system test, bus 1 is encoder input, bus 2 is decoder output

Appendix B. Layout simulation

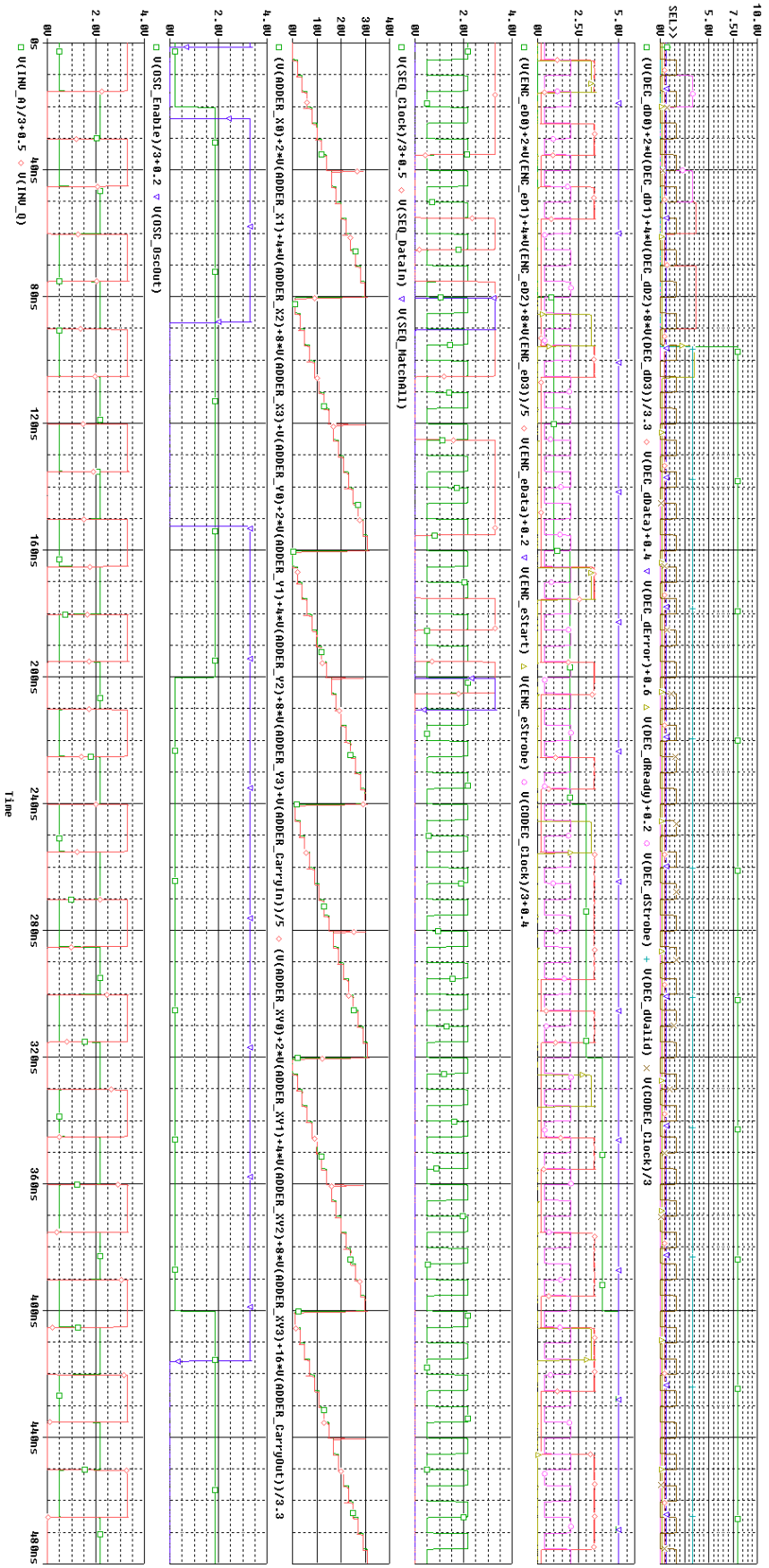
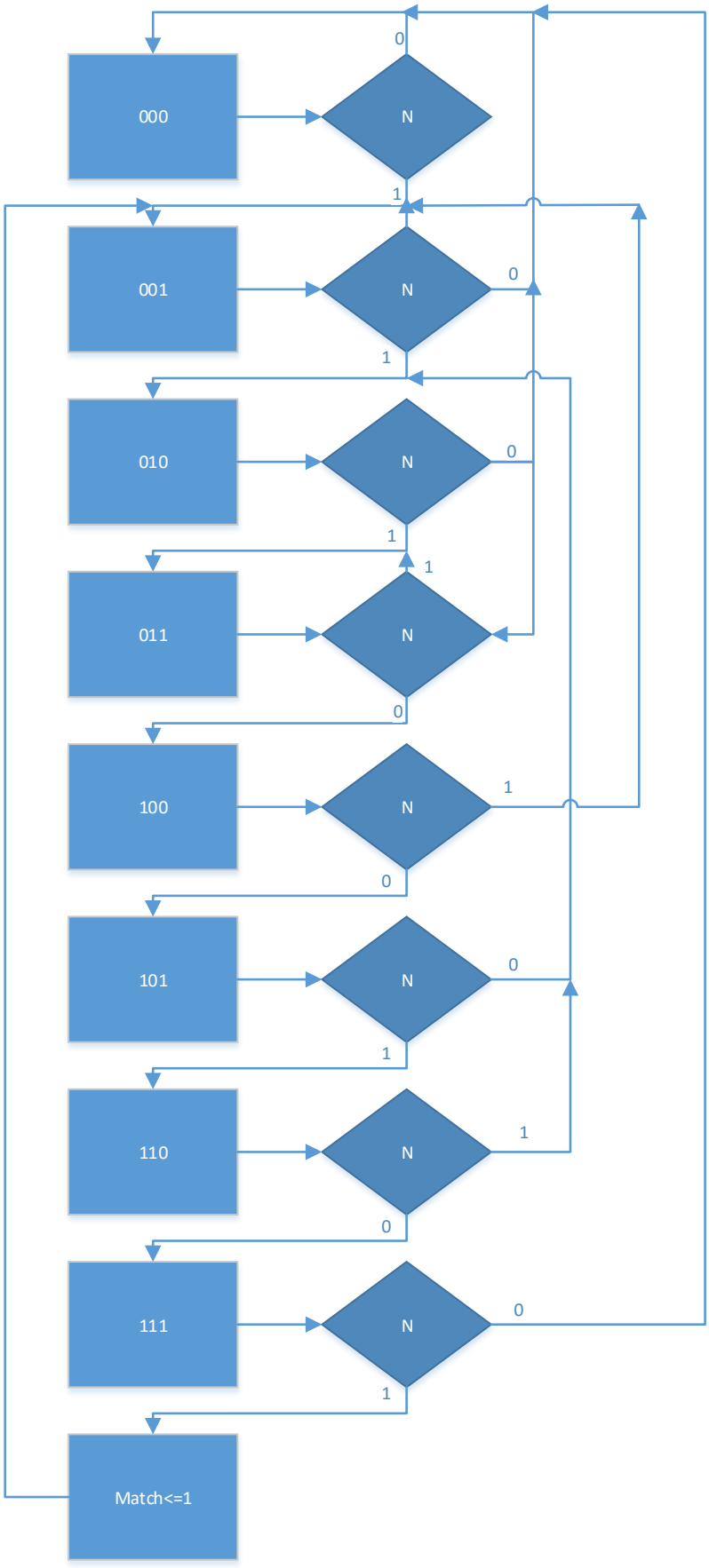


Figure 3 Layout simulation using OrCAD

Appendix C. Sequencer ASM chart



Appendix D. Ring oscillator schematic design

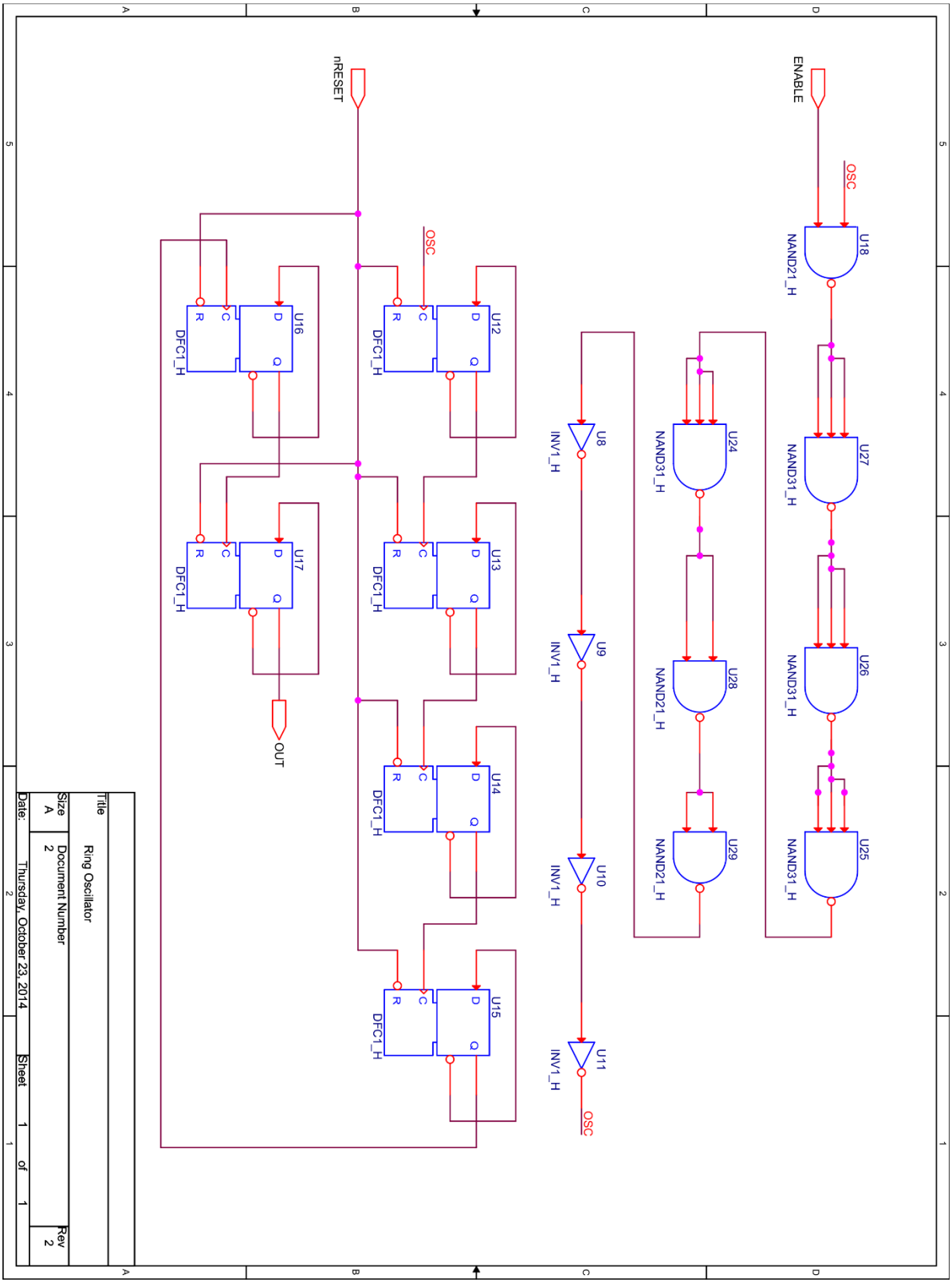


Figure 4 Ring oscillator schematic design

Appendix E. Encoder ALU schematic design

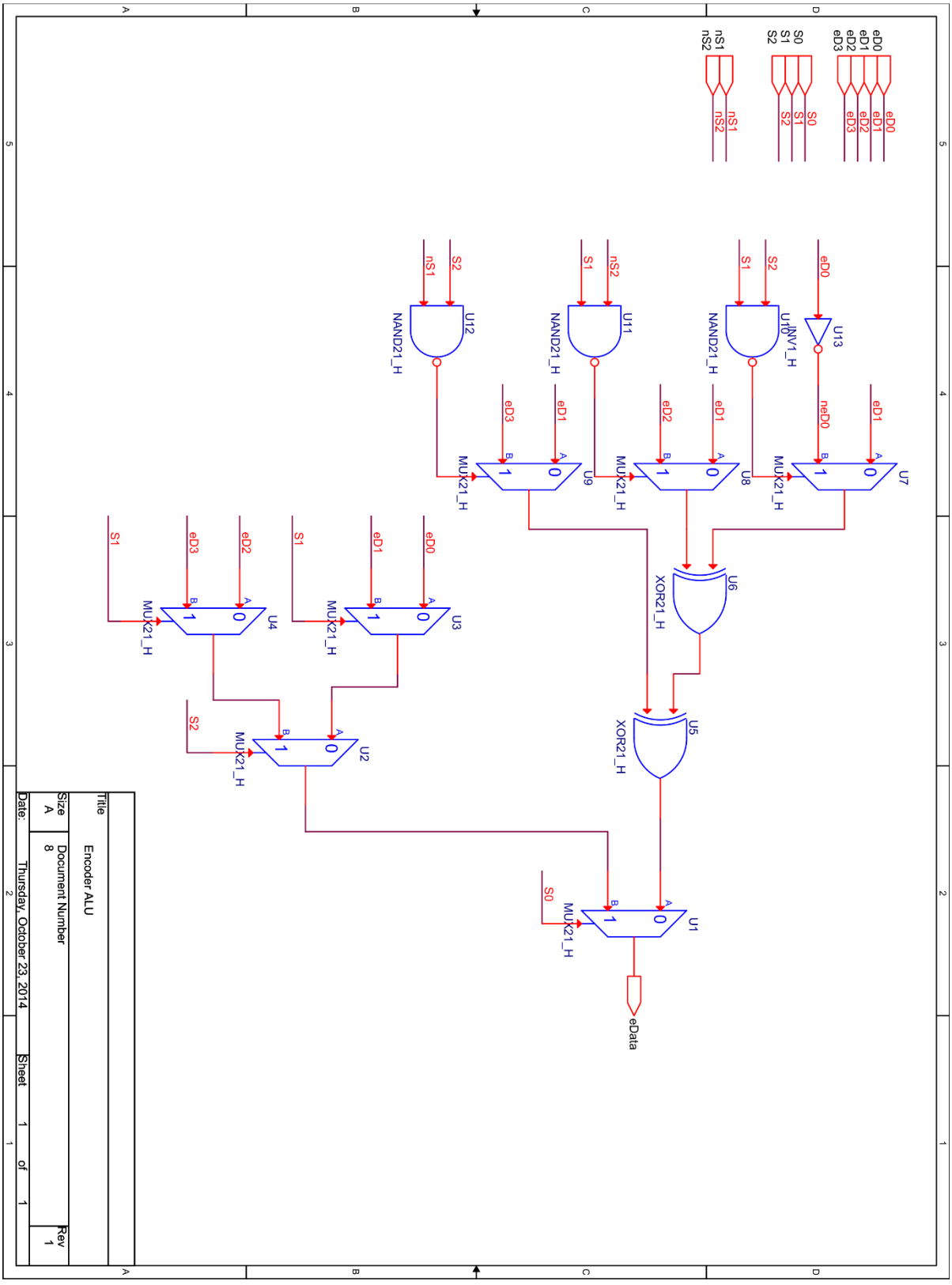


Figure 5 Encoder ALU schematic design, S[0:2] and nS[1:2] come from state machine, eD[0:3] come from input data

Listing 1: ./AVR/Codec/main.cpp

```cpp
#include <avr/io.h>
#include <stdio.h>
#include <stdint.h>
#include <util/delay.h>
#include <tft.h>
#include <colours.h>

// PORT B
#define COM_CLK   _BV(0)
//#define COM_RESET _BV(1)
#define COM_DATA  _BV(1)
#define ENC_START _BV(4)
#define DEC_READY _BV(5)
#define DEC_VALID _BV(6)
#define DEC_ERROR _BV(7)

// PORT D
#define ENC_DBIT  (ENC_D0 | ENC_D1 | ENC_D2 | ENC_D3)
#define ENC_D0    _BV(0)
#define ENC_D1    _BV(1)
#define ENC_D2    _BV(2)
#define ENC_D3    _BV(3)
#define DEC_DBIT  (DEC_D0 | DEC_D1 | DEC_D2 | DEC_D3)
#define DEC_DATA  ((PIND & DEC_DBIT) >> 4)
#define DEC_D0    _BV(4)
#define DEC_D1    _BV(5)
#define DEC_D2    _BV(6)
#define DEC_D3    _BV(7)

#define DATA_WAITING  0xFF
#define DATA_INVALID  0x7F
#define DATA_ERROR _BV(6)
#define AVERAGE   200
#define RATE      0.01

using namespace colours::b16;

tft_t tft;

class decoder_t
{
public:
    decoder_t(void) : errCount(0), cnt(0), correct(0), errRate(0) {}
    void check(const uint8_t data, const uint8_t dec);
    uint8_t recv(void);
    uint16_t errors(void) const {return errCount;}
    uint16_t count(void) const {return cnt;}
    double errorRate(void) const {return (double)errCount / cnt;}

private:
    uint16_t errCount, cnt;
    uint16_t correct;
    double errRate;
} decoder;

uint16_t noiseCount = 0, dataCount = 0;

void noise(float rate)
{
    if ((rand() & 0xFFFF) < (rate * 0xFFFF)) {
        PORTB |= COM_DATA;
        noiseCount++;
    } else
        PORTB &= ~COM_DATA;
}

bool enc_pool(uint8_t data)
{
    PORTB |= COM_CLK;
    PORTB &= ~COM_CLK;
    static uint8_t cnt = 0;
    if (data == DATA_WAITING)
        PORTB &= ~ENC_START;
    else {
        PORTD = (PORTD & ~ENC_DBIT) | (data & ENC_DBIT);
        PORTB |= ENC_START;
        cnt = 8 - 1;
    }
    if (cnt == 0)
        return true;
    cnt--;
    return false;
}

uint8_t decoder_t::recv(void)
{
    uint8_t status = PINB;
    if (!(status & DEC_READY))
        return DATA_WAITING;
    cnt++;
    if (!(status & DEC_VALID))
        return DATA_INVALID;
    uint8_t data = DEC_DATA;
    if (status & DEC_ERROR)
        data |= DATA_ERROR;
    return data;
}

void decoder_t::check(const uint8_t data, const uint8_t dec)
{
    bool error = dec == DATA_INVALID || (dec & ~DATA_ERROR) != data;
    errCount += error;
    errRate = (errRate * (AVERAGE - 1) + error) / AVERAGE;
}

void init(void)
{
    DDRB = COM_CLK | COM_DATA | ENC_START;
    PORTB = DEC_READY | DEC_VALID | DEC_ERROR;
    DDRD = ENC_DBIT;
    PORTD = DEC_DBIT;

    tft.init();
    tft.setOrient(tft.Portrait);
    tft.setBackground(Black);
    tft.setForeground(0x667F);
    tft.clean();
    stdout = tftout(&tft);
    tft.setBGLight(true);
}

int main(void)
{
    init();

    tft.clean();
    tft.setZoom(1);
    puts("Hamming encoder & decoder test");

    tft.setZoom(2);
    bool newData = true;
loop:
    uint8_t data = 0;
    while (data < 0x10) {
        noise(RATE);
        newData = enc_pool(newData ? data : DATA_WAITING);
        data += newData;
        dataCount += newData;
        uint8_t dec = decoder.recv();
        if (dec == DATA_WAITING)
            continue;
        decoder.check((data - 1) & 0x0F, dec);
    }
    tft.setY(0);
    tft.setForeground(Red);
    puts("Bit error rate:");
    tft.setForeground(Green);
    printf("%6.2f%%\n", (double)noiseCount * 8 / dataCount);
    tft.setForeground(Red);
    puts("Data error rate:");
    tft.setForeground(Green);
    printf("%6.2f%%\n", decoder.errorRate() * 100.0);
    tft.setForeground(Red);
    puts("Data count:");
    tft.setForeground(Green);
    printf("%u\n", dataCount);
    tft.setForeground(Red);
    puts("Data error count:");
    tft.setForeground(Green);
    printf("%u\n", decoder.errors());
    goto loop;

    return 1;
}
```

Listing 2: ./test/Adder/main.cpp

```cpp
#include <stdio.h>
#include <stdint.h>

void printBinary(uint32_t v, int len)
{
    char str[32];
    str[len] = '\0';
    while (len--) {
        str[len] = v % 2 ? '1' : '0';
        v >>= 1;
    }
    printf(str);
}

int main(void)
{
    puts("# 4 bit adder\n"
    "<PinDef>\n"
    "# CarryIn, A[3:0], B[3:0], CarryOut, Q[3:0]\n"
    "A3, A7, A6, A5, A4, A11, A10, A9, A8, Q7, Q6, Q5, Q4, Q3\n"
    "</PinDef>\n\n"
    "<TestVector>");

    for (uint8_t c = 0; c < 2; c++)
        for (uint8_t a = 0; a < 16; a++)
            for (uint8_t b = 0; b < 16; b++) {
                printBinary(c, 1);
                putchar(' ');
                printBinary(a, 4);
                putchar(' ');
                printBinary(b, 4);
                putchar(' ');

                printBinary((a + b + c) / 0x10, 1);
                putchar(' ');
                printBinary(a + b + c, 4);
                putchar('\n');
            }
    puts("</TestVector>");

    return 0;
}
```

Listing 3: ./test/Combine/main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>

#define MAX 8

using namespace std;

bool clockExist = false;

struct pin_t {
    string name;
    enum Values {Low, High, Clock, Unknown} value;

    static Values toValue(const char c);
```

```
};
vector<pin_t> pins;

struct vec_t {
    vec_t(void) : clocked(false), finished(false) {}
    bool open(const char *path);
    bool readHeader(const bool rebuild = false);
    bool printVector(void);
    void printVector(const string &str);
    bool accept(const string &str);

    struct pin_t {
        pin_t(void) : dup(false) {}

        string name;
        bool dup;
    };
    vector<pin_t> pins;
    ifstream fs;
    string buffer, last;
    bool clocked, finished;
} vec[MAX];

string trim(const string &str)
{
    istringstream ss(str);
    string res, tmp;
    while (ss >> tmp)
        if (res.empty())
            res = tmp;
        else
            res.append(" " + tmp);
    return res;
}

string nospace(const string &str)
{
    istringstream ss(str);
    string res, tmp;
    while (ss >> tmp)
        res.append(tmp);
    return res;
}

bool checkDupPin(const string &name)
{
    for (vector<pin_t>::iterator it = pins.begin(); it != pins.end(); it++)
        if (name == it->name)
            return true;
    return false;
}

void setPin(const string &name, pin_t::Values v)
{
    for (vector<pin_t>::iterator it = pins.begin(); it != pins.end(); it++)
        if (name == it->name)
            it->value = v;
}

pin_t::Values pin_t::toValue(const char c)
{
    switch (c) {
    case '0':
        return Low;
    case '1':
        return High;
    case 'C':
        return Clock;
    default:
        return Unknown;
    }
}

bool vec_t::open(const char *path)
{
    fs.open(path);
    if (!fs.good()) {
        cerr << "Error opening file: " << path << endl;
        return false;
    }
    return true;
}

bool vec_t::readHeader(const bool rebuild)
{
    string str;
    fs.seekg(0);
read:
    getline(fs, str);
    if (fs.eof() || str.empty())
        goto read;
    if (str.at(0) == '#') {
        if (rebuild)
            cout << str << endl;
        goto read;
    }
    if (str == "<PinDef>")
        goto read;
    if (str == "</PinDef>")
        return true;
    if (!rebuild)
        goto read;
    istringstream iss(str);
    string token;
readPin:
    if (!getline(iss, token, ','))
        goto read;
    vec_t::pin_t pin;
    pin.name = trim(token);
    if (!(pin.dup = checkDupPin(pin.name))) {
        ::pin_t p;
        p.name = pin.name;
        ::pins.push_back(p);
    }
    pins.push_back(pin);
    goto readPin;
}

void vec_t::printVector(const string &str)
{
    int i = 0;
    for (vector<pin_t>::iterator it = pins.begin(); it != pins.end(); it++, i++)
        if (!it->dup) {
            cout << str.at(i);
            ::pin_t::Values v = ::pin_t::toValue(str.at(i));
            setPin(it->name, v);
            if (v == ::pin_t::Clock) {
                clocked = true;
                clockExist = true;
            }
        }
}

bool vec_t::accept(const string &str)
{
    // Clock required
    bool clk = false;
    int i = 0;
    for (vector<pin_t>::iterator it = pins.begin(); it != pins.end(); it++, i++)
        if (it->dup) {
            for (vector< ::pin_t >::iterator git = ::pins.begin(); git != ::pins.end();
                    git++)
                if (it->name == git->name && git->value != ::pin_t::Unknown &&
                        ::pin_t::toValue(str.at(i)) != git->value)
                    return false;
        } else if (str.at(i) == 'C')
            clk = true;
    if (clockExist && clk)
        return false;
    return true;
}

bool vec_t::printVector(void)
{
    if (!buffer.empty()) {
        if (!accept(buffer))
            printVector(last);
        else {
            printVector(buffer);
            buffer.clear();
        }
        return !finished;
    }
    string str;
read:
    getline(fs, str);
    if (fs.eof() || str == "</TestVector>") {
        finished = true;
        if (!clocked) {
            readHeader(false);
            goto read;
        }
        printVector(last);
        return false;
    }
    if (str.empty() || str.at(0) == '#' || str == "<TestVector>")
        goto read;
    str = nospace(str);
    if (!accept(str)) {
        printVector(last);
        buffer = str;
        return !finished;
    }
    last = str;
    printVector(last);
    finished = finished || fs.eof();
    return !finished;
}

int main(int argc, char *argv[])
{
    if (argc <= 1) {
        clog << "Usage: " << argv[0] << " [.vec files...]" << endl;
        return 0;
    } else if (argc > 1 + MAX) {
        cerr << "Too many files!" << endl;
        return 1;
    }
    const int cnt = argc - 1;
    cout << "<PinDef>" << endl;
    for (int i = 0; i < cnt; i++) {
        if (i)
            cout << endl;
        if (!vec[i].open(argv[1 + i]))
            return 1;
        if (!vec[i].readHeader(true))
            return 1;
    }
    for (vector<pin_t>::iterator it = pins.begin(); it != pins.end(); it++)
        if (it == pins.begin())
            cout << it->name;
        else
            cout << ", " << it->name;
    cout << endl;
    cout << "</PinDef>" << endl;

    cout << "<TestVector>" << endl;
printVector:
    clockExist = false;
    for (vector<pin_t>::iterator it = pins.begin(); it != pins.end(); it++)
        it->value = pin_t::Unknown;
    bool pnt = false;
    for (int i = 0; i < cnt; i++) {
        if (i)
            cout << " ";
        pnt = vec[i].printVector() || pnt;
    }
    cout << endl;
    if (pnt)
        goto printVector;
    cout << "</TestVector>" << endl;
    return 0;
```

```
}
```

Listing 4: ./test/Decoder/main.cpp

```
#include <stdio.h>
#include <stdint.h>

#define PREV  1
#define DELAY 0

#define DEC_ERRBIT (1 << 7)
#define DEC_ERROR(d)  (d & DEC_ERRBIT)
#define DEC_VALID(d)  (d != 0xFF)
#define DEC_TRANSERRBIT (1 << 6)
#define DEC_TRANSERR(d) (d & DEC_TRANSERRBIT)

void printBinary(uint32_t v, int len)
{
    char str[32];
    str[len] = '\0';
    while (len--) {
        str[len] = v % 2 ? '1' : '0';
        v >>= 1;
    }
    printf(str);
}

uint8_t decoder(uint8_t data)
{
    bool a, b, c, d;
    a = ((data >> 7) & 0x01) ^ ((data >> 5) & 0x01) ^ \
        ((data >> 1) & 0x01) ^ ((data >> 0) & 0x01);
    b = ((data >> 7) & 0x01) ^ ((data >> 3) & 0x01) ^ \
        ((data >> 2) & 0x01) ^ ((data >> 1) & 0x01);
    c = ((data >> 5) & 0x01) ^ ((data >> 4) & 0x01) ^ \
        ((data >> 3) & 0x01) ^ ((data >> 1) & 0x01);
    d = ((data >> 7) & 0x01) ^ ((data >> 6) & 0x01) ^ \
        ((data >> 5) & 0x01) ^ ((data >> 4) & 0x01) ^ \
        ((data >> 3) & 0x01) ^ ((data >> 2) & 0x01) ^ \
        ((data >> 1) & 0x01) ^ ((data >> 0) & 0x01);
    if (d && (!a || !b || !c))
        return 0xFF;
    data = (((data >> 7) & 0x01) << 3) | (((data >> 5) & 0x01) << 2) | \
           (((data >> 3) & 0x01) << 1) | (((data >> 1) & 0x01) << 0);
    if (d)
        return data;
    else if (!a && !b && !c)
        return (data ^ 0x01) | DEC_ERRBIT;
    else if (a && !b && !c)
        return (data ^ 0x02) | DEC_ERRBIT;
    else if (!a && b && !c)
        return (data ^ 0x04) | DEC_ERRBIT;
    else if (!a && !b && c)
        return (data ^ 0x08) | DEC_ERRBIT;
    return data | DEC_ERRBIT;
}

int main(void)
{
    puts("# Hamming decoder\n"
    "<PinDef>\n"
    "# Clock, nReset, Strobe, DataIn, D3, D2, D1, D0, Ready, Valid, Error\n"
    "A15, A16, A22, A23, Q21, Q20, Q19, Q18, Q17, Q22, Q23\n"
    "</PinDef>\n\n"
    "<TestVector>\n"
    "0000 0000 000\n"
    "C000 0000 000\n");

    int prev = -1, prevprev = -1;
    for (int data = 0; data < 0x100 + 1; data++) {
        uint8_t dec = decoder(data);
        for (int i = 0; i < 8 + DELAY; i++) {
            if (PREV) {
                if (i == 8 - PREV && data < 0x100)
                    prev = dec;
            } else if (i == DELAY)
                prev = prevprev;
            bool in = i < 8 ? (data >> i) % 2 : 0;
            printf("C1%u%u ", i == 0 && data < 0x100, in);
            if (DEC_VALID(prev))
                printBinary(prev == -1 ? 0 : prev, 4);
            else
                printf("XXXX");
            bool ready = 0, valid = 0, error = 0;
            if (prev != -1) {
                ready = PREV ? (data < 0x100 ? i == 8 - PREV : 0) : i == DELAY;
                valid = DEC_VALID(prev);
                error = ready ? DEC_ERROR(prev) : 0;
            }
            printf(" %u%u%u", ready, valid, error);
            putchar('\n');
        }
        prevprev = dec;
        putchar('\n');
    }
    puts("C100 XXXX 0X0"); // Idle state
    puts("0100 XXXX 0X0"); // Idle state
    puts("</TestVector>");

    return 0;
}
```

Listing 5: ./test/Encoder/main.cpp

```
#include <stdio.h>
#include <stdint.h>

#define DELAY 0

void printBinary(uint32_t v, int len)
{
    char str[32];
    str[len] = '\0';
    while (len--) {
        str[len] = v % 2 ? '1' : '0';
        v >>= 1;
    }
    printf(str);
}

uint8_t encoder(uint8_t num)
{
    uint8_t exp = ((num & 0x08) << 4) | ((num & 0x04) << 3) | ((num & 0x02) << 2) |
            ((num & 0x01) << 1);
    exp |= ((num & 0x02) << 5) ^ ((num & 0x04) << 4) ^ ((num & 0x08) << 3);
    exp |= ((~num & 0x01) << 4) ^ ((num & 0x02) << 3) ^ ((num & 0x04) << 2);
    exp |= ((~num & 0x01) << 2) ^ ((num & 0x02) << 1) ^ ((num & 0x08) >> 1);
    exp |= (~num & 0x01) ^ ((num & 0x04) >> 2) ^ ((num & 0x08) >> 3);
    return exp;
}

int main(void)
{
    puts("# Hamming encoder\n"
    "<PinDef>\n"
    "# Clock, nReset, Start, D3, D2, D1, D0, Strobe, DataOut\n"
    "A15, A16, A17, A21, A20, A19, A18, Q15, Q16\n"
    "</PinDef>\n\n"
    "<TestVector>\n"
    "000 0000 00\n"
    "C00 0000 00\n");

    for (int n = 0; n < 0x10; n++) {
        for (int i = 0; i < 8 + DELAY; i++) {
            printf("C1%u ", i == 0);
            printBinary(n, 4);
            int e = encoder(n);
            bool o = i < DELAY ? 0 : ((e >> (i - DELAY)) & 0x01);
            printf(" %u%u\n", i == DELAY ? 1 : 0, o);
        }
        putchar('\n');
    }
    puts("C10 0000 0X");  // Idle state
    puts("010 0000 0X");  // Idle state
    puts("</TestVector>");

    return 0;
}
```

Listing 6: ./test/Sequencer/main.cpp

```
#include <stdio.h>
#include <stdint.h>

#define TEAMI  0b11100001
#define TEAMJ  0b11100101
#define TEAMK  0b11101001
#define TEAMM  0b11110001
#define TEAMN  0b11110101

const uint8_t test = TEAMJ;

void printBinary(uint32_t v, int len)
{
    char str[32];
    str[len] = '\0';
    while (len--) {
        str[len] = v % 2 ? '1' : '0';
        v >>= 1;
    }
    printf(str);
}

bool match(bool in)
{
    static uint8_t match = 0;
    match <<= 1;
    match |= in ? 1 : 0;
    return match == test;
}

void check(uint8_t data, uint8_t bits)
{
    for (uint8_t i = 0; i < bits; i++) {
        bool in = data & 0x80;
        data <<= 1;
        printf("C1%u", in);
        printf(" %u\n", match(in));
    }
}

int main(void)
{
    printf("# Sequencer: ");
    printBinary(test, 8);
    puts("\n<PinDef>\n"
    "# Clock, nReset, DataIn, MatchAll\n"
    "A12, A13, A14, Q12\n"
    "</PinDef>\n\n"
    "<TestVector>\n"
    "000 0\n"
    "C00 0\n");

    check(test, 8);
    putchar('\n');
    for (uint8_t i = 1; i <= 8; i++) {
        check(test, i);
        check(test, 8);
        putchar('\n');
    }
#ifdef TEST_ALL
    for (int i = 0; i < 256; i++) {
        check(i, 8);
        putchar('\n');
    }
#endif
    puts("000 0\n");  // Idle state
    puts("</TestVector>");

    return 0;
}
```