App Note 006

Applying the AN10E40
Magnetic Stripe Read Head Amplifier

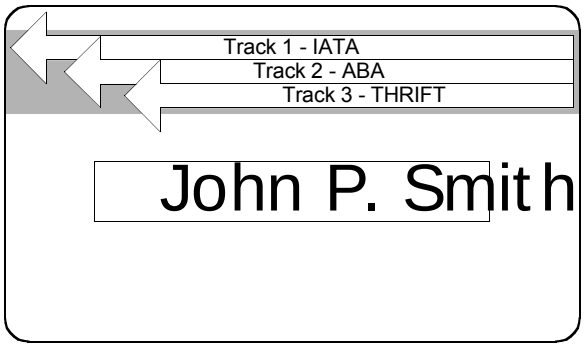
Introduction

Every now and then a technology comes along that's easy to use, robust and adapts to many applications. Magnetic data stripes are one such technology. The obvious example is the credit card. But there are many other examples including: paper airline tickets, shopping club cards, video rental store ID cards, driver licenses, vending machine debit cards, and the list goes on.

This application note briefly describes how an Anadigm Field Programmable Analog Array (FPAA) can be used to construct a complete magnetic stripe read amplifier / decoder and how it can be easily interfaced to a host processor. The note also describes the benefits of programmable analog verses fixed function in such an application.

The Standard Card

"The beautiful thing about standards is that there are so many to choose from." How true. There are loads of standards to choose from when discussing magnetic stripe technologies, and I shall not endeavor to enumerate them all here. Suffice it to say that the most often referenced specification is ISO/IEC-7811. I'll just capture the highlights here with the understanding that we are using a read head and housing intended for plastic card magnetic stripe read applications. There are many quality suppliers of such, in this instance Magtek products were utilized.



For this discussion, the figure to the left represents the back of a typical plastic credit card. With the magnetic stripe on top as shown, the first (top) track is encoded in a format established by the International Air Transport Association, this track usually has your name encoded on it.

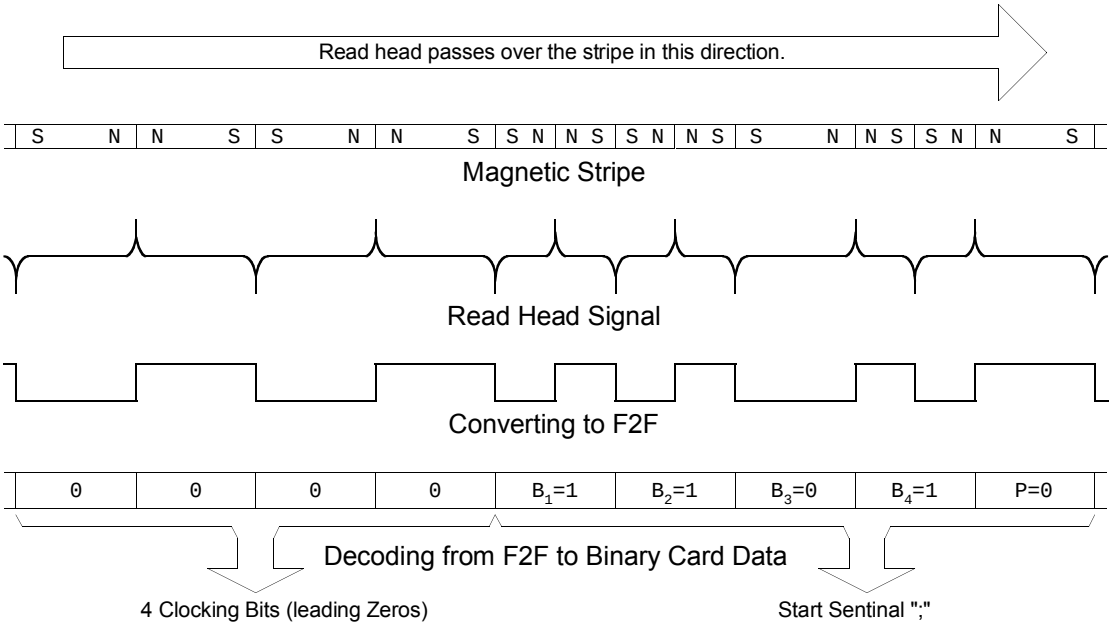
The second track (and this the real workhorse of the bunch) is in a format defined by the American Bankers Association. It contains your credit card number. The third track format is called THRIFT, and was originally intended for use with Automatic Teller Machines (ATMs). Unlike the read-only tracks 1 and 2, the THRIFT track was intended for read and write applications. This never really caught on and the track is not often used, except in those applications where the requirement to write back data is part of the design. Typical examples include copy machine and vending machine debit cards.

Viewing the card as shown above, data is encoded from right to left. The encoding always starts with the least significant bit. (The right most bit is the LSB of the first character for the stripe.) The definition for each track is different and is summarized in the table below.

	Bits per Character	Bits Per Inch
Track 1 - IATA	Alpha - 6 Data + 1 Parity (odd)	210
Track 2 - ABA	BCD - 4 Data + 1 Parity (odd)	75
Track 3 - Thrift	BCD - 4 Data + 1 Parity (odd)	210

From Mag Stripe to ASCII Type…

There are a number of steps required to get from the magnetic read head signal to ASCII data digestible by most processors. The following diagram shows these translation steps as we move from top to bottom.



The Magnetic Stripe
Understanding the conversion process begins with knowing how data is stored on the card. Imagine a long chain of bar magnets, some are one unit in length (and always found in pairs) all the rest are two units long. Further, each of these bar magnets is arranged contrary to the state they would prefer, such that North always touches North, and South always touches South. In these areas where like poles are abutted, there are large concentrations of magnetic flux lines.

The Read Head Signal
The magnetic stripe is swiped passed the read head. A current will be induced as each of these regions of high flux concentration passes. N-N will induce a current in one direction, and S-S will induce a current in the opposite direction. As it turns out, the polarity of the read head signal doesn't matter. It is only the spacing in time of the transitions that is relevant to converting to an F2F waveform.

F2F or Aiken Biphase
For now, we'll put off the discussion of just how to create the F2F waveform from the read head signal and instead focus only on what F2F encoding is. Also known as Aiken Biphase, F2F encoding uses the space between flux transitions to encode a bit. A two unit bar magnet represents a Zero, and a pair of one unit bars represents a One.

To put it another way, each bit occupies the same physical length on the stripe. A bit with an 'extra' flux transition in the middle of its length is called a One. Again, the polarity of these transitions is arbitrary, it is the relative space between transitions that signifies a One or a Zero.

Converting from Card Data to ASCII
Each track begins and ends with "clocking bits". Clocking bits are nothing more than a string of Zeros. As it turns out, a manual card swipe is a fairly constant speed event. Thus, a bit period from the beginning of a swipe is very close to a bit period at the end of a swipe. The "clocking bits" are examined for length (the bit period) and set the pace of the particular swipe. A Zero is recognized if the time elapsed between flux transitions is close to a bit period. A One is recognized if the time elapsed between flux transitions is close to half a bit period.

For processors equipped with a timer peripheral, it is possible to connect the FPAA output (which will be toggling between 0 and 5 V during a card swipe) to the timer port's input pin. Interrupts will be generated with each transition and storing the timer values into an array is all that is required of the interrupt service routine. A quick bit of post processing on the array of values easily deciphers the Ones and Zeros.

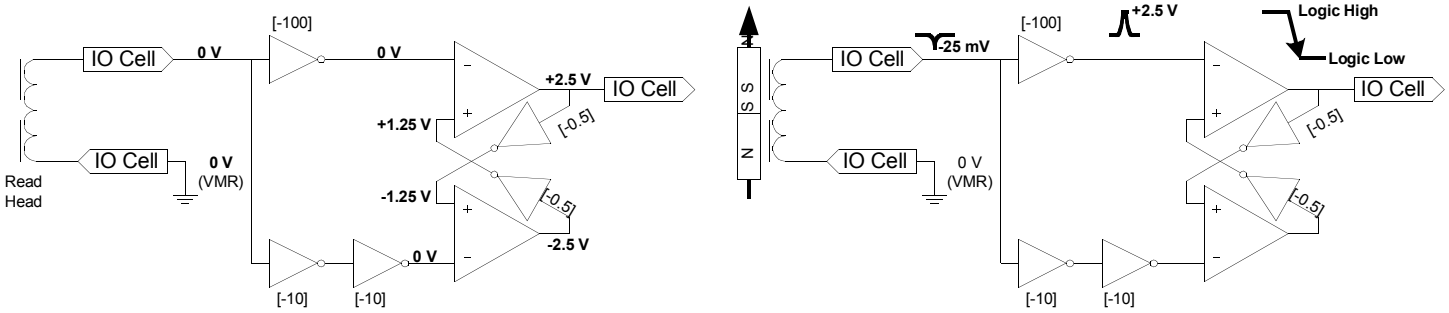
For lower cost processors, you'll have to instead connect the FPAA output to a general purpose input pin on the processor. Once a first transition is detected (usually as an interrupt), the processor then has to completely devote itself to keeping track of time, polling the input pin at regular intervals, and recording when the transitions occur. Here again, post processing of the array of time stamps will decipher the Ones and Zeros.

Keep in mind that the card may have been swiped backwards. Flipping the data around is easily handled in software.

Now that you have a string of Ones and Zeros, you need to recognize where the data started, and how to interpret it. For an introduction to this part of the process, please see this note's Addendum.

Applying the AN10E40 as a Magnetic Read Head Amplifier and F2F Generator

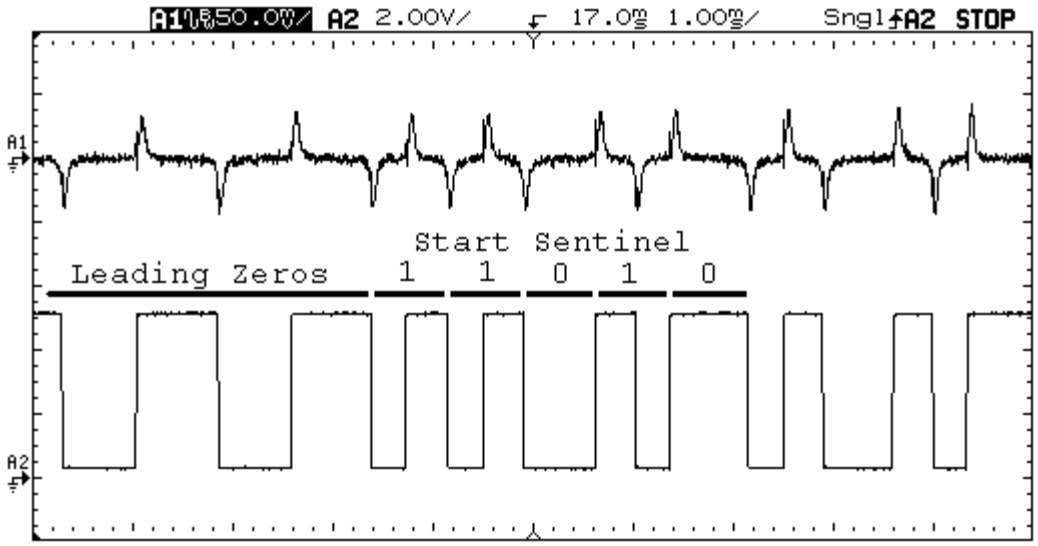
Only G01 gain stages and C02 comparators are required. The circuit demonstrated here is a sort of an "analog S-R flip-flop". The comparators and their feedback paths form a bi-stable circuit. The circuit will stabilize in only two states.



Looking at the circuit diagram on the left. Voltages are shown bold, and amplifier gains are shown in square brackets. The amplifiers are shown with a bubble on their outputs just to serve as a visual reminder that these are all inverting amplifiers. Assume there is no card present and that all node voltages are as shown. (A reminder: All analog signal processing within the FPAA is done with respect to Voltage Mid-Rail (VMR), and by convention this is labeled as 0 V. VMR is actually 2.5 V above the chip's ground… so for the above left diagram, a processor would see the output of +2.5 V as +5 volts (logic high) and an output of -2.5 V as 0 volts (logic low).)

Examine the node voltages for a moment and you'll see that the circuit is in a stable state. The output is a logic high.

Now consider what happens when a card is swiped passed the read head. For the example shown above right, a S-S flux field passed over gap in the read head. This induced a small signal negative going spike. The high gain input stages convert this small spike into a full rail signal. (Any input signal with an amplitude of greater than 25 mV will simply result in a clipped signal as shown.) This positive pulse presented to the negative terminal of the top comparator will cause it to change state as shown by the falling output waveform. Just the opposite is happening on the lower comparator as the entire circuit transitions to its other stable state. The output is now a logic low.



The figure above is a scope shot of the circuit in action. In this particular case, a read head was interfaced directly to an AN10E40. The top trace shows the raw read head input… noise and all. The bottom trace shows the circuit's output

signal (a 0 - 5V F2F logic waveform). No other components were used. Nothing to it; its that easy with programmable analog.

The circuit as shown above consumed only 9 of the 20 CAB's available within an AN10E40. In point of fact, the two series amplifiers in the bottom leg of the circuit really aren't required, provided that the bottom comparator's negative terminal is instead connected to VMR, and this reduces the CAB consumption by two. There is plenty of room in an AN10E40 for a second read channel with room to spare.

Benefits of Using Programmable Analog

The circuit described is simple enough and can of course be constructed using twenty or so standard components. So why programmable analog?

The obvious answer is component count. There were no extra components of any kind used in this application. The read head was interfaced directly to the FPAA. However, the real advantages don't become obvious until you take the circuit out of the lab and into the harsh light of the real world.

In the real world credit card stripe readers constructed without programmable analog aren't all they should be. You've seen evidence of this so many times that you have probably become numb to it. Take a moment and recall all the cashier "swipe" techniques you have witnessed. The "slow and deliberate" swipe. The "fast swipe". The "run it backwards" swipe. The "run it back and forth and back and forth" technique (one of my personal favorites). The "clean the card and try it again swipe". The "wrap the card in a plastic bag and try it again swipe". The "try the other card reader " technique. And then of course there it the ultimate contingency plan where the cashier holds the card up to the light just right so the badly worn raised numbers can be recognized and typed in by hand (usually while sharing a disgusted look with you).

Dirt, wear, temperature, and exposure to unintentional magnetic fields all conspire to render magnetic stripe cards difficult to read. Likewise, dirt, wear and corrosion work to render the read heads less and less efficient. So how can programmable analog fix the situation? The host processor can download a complete new circuit configuration to an AN10E40 in under 125 microseconds. So when a swipe fails, you can adjust the gain of the input amplifiers in just a fraction of a second. In fact, in a carefully designed system, you can adjust the read amplifiers during the leading clocking bits! A second swipe will not be necessary.

The circuit can also be adjusted for reading cards with differing magnetic stripe coercivity. While usually not important for read heads, adjusting write head drive signals is essential to accommodate such coercivity differences.

These are some the advantages of applying programmable analog just in standard card applications. In custom applications, there are even more advantages to using programmable analog. For high security applications, the FPAA can be configured to handle different data encoding techniques including tone generation and decoding. Read and write functions and more.

Addendum - Credit Card Track Encoding and Data Standards

There are a lot of steps involved in recovering data off a simple magnetic stripe. So far we have covered read head amplification, generation of an F2F waveform and its interpretation into a string of ones and zeros. The next step in the recovery process is to convert the binary data into character data, usually ASCII. The rules for interpreting the data vary with which track you are reading.

Track Data Encoding and Parameters

Track	Standard	BPI	Character Type	Number of Characters	Encoding Scheme	Comments
1	IATA	210	Alphanumeric	79	6 Bits + Odd Parity	Read Only
2	ABA	75	BCD	40	4 Bits + Odd Parity	Read Only
3	Thrift	210	BCD	107	4 Bits + Odd Parity	Read & Write (uncommon usage)

Common Track Layouts
Common to every track are the "clocking bits". Clocking bits are found at both ends of every track. Clocking bits are short string of zeros intended to give self clocking card readers a chance to establish the swipe speed and set a bit cell time duration. In our application, the microprocessor will examine the clocking bit periods as a first step in decoding the binary data.

Also common to every track are the Start Sentinel (SS), End Sentinel (ES) and Logitudnal Redundancy Check (LRC) characters. On Track 1, the SS and ES characters are % and ?, respectively. On Tracks 2 and 3, the SS and ES characters are ; and ?, respectively. Field Separators (FS) are common to Tracks 1 and 2.

If at first you don't recognize a Start Sentinel, then chances are the card was swiped backwards. You will have to adjust your decoding algorithm accordingly.

The LRC is the sum of all previous B_n bits for all the previous characters on the stripe (overflow is ignored). The odd Parity bit associated with each character will only flag a problem an odd number of bits (like 1) were in error for that character. It is unlikely that there will be 2 bit errors in a single character, but if there is, the odd parity checking will not recognize the event. The job of the LRC then is to add one more layer of error checking for the data stream associated with the entire track.

Track 1 Specific Layout

zeros | SS | FC | Primary Acct (19 characters max) | Name (26 characters max) | FS | other data | ES | LRC | zeros

Track 2 Specific Layout

zeros | SS | Primary Acct (19 characters max) | FS | other data | ES | LRC | zeros

Track 3
There are too many non-standard uses of Track 3 to enumerate here. Suffice it to say, it is largely an abandoned track and consequently is where most specialized or custom systems will encode data. The original intent was to use this track as a Read/Write track and actually carry encrypted information about your bank account balance here. This allowed non-networked automated teller machines to dispense money without really knowing for sure what your actual balance was at the moment of the withdrawal. It didn't take the banking industry very long to realize that it is far better to instead network together ATMs and abandon the idea of dynamic data on your ATM card. Track 3 is now an orphan.

Track 1 Character Set - 6 Bit Alpha with Odd Parity

Character	P	B_6	B_5	B_4	B_3	B_2	B_1	ASCII
Space	1	0	0	0	0	0	0	20
(undefined)								21
								22
(OG)	1	0	0	0	0	1	1	23
$	0	0	0	0	1	0	0	24
%(SS)	1	0	0	0	1	0	1	25
								26
								27
(	0	0	0	1	0	0	0	28
)	1	0	0	1	0	0	1	29
								2A
								2B
								2C
-	0	0	0	1	1	0	1	2D
.	0	0	0	1	1	1	0	2E
/	1	0	0	1	1	1	1	2F
0	0	0	1	0	0	0	0	30
1	1	0	1	0	0	0	1	31
2	1	0	1	0	0	1	0	32
3	0	0	1	0	0	1	1	33
4	1	0	1	0	1	0	0	34
5	0	0	1	0	1	0	1	35
6	0	0	1	0	1	1	0	36
7	1	0	1	0	1	1	1	37
8	1	0	1	1	0	0	0	38
9	0	0	1	1	0	0	1	39
								3A
								3B
								3C
=	1	0	1	1	1	0	1	3D
								3E
?(ES)	0	0	1	1	1	1	1	3F

Character	P	B_6	B_5	B_4	B_3	B_2	B_1	ASCII
								40
A	1	1	0	0	0	0	1	41
B	1	1	0	0	0	1	0	42
C	0	1	0	0	0	1	1	43
D	1	1	0	0	1	0	0	44
E	0	1	0	0	1	0	1	45
F	0	1	0	0	1	1	0	46
G	1	1	0	0	1	1	1	47
H	1	1	0	1	0	0	0	48
I	0	1	0	1	0	0	1	49
J	0	1	0	1	0	1	0	4A
K	1	1	0	1	0	1	1	4B
L	0	1	0	1	1	0	0	4C
M	1	1	0	1	1	0	1	4D
N	1	1	0	1	1	1	0	4E
O	0	1	0	1	1	1	1	4F
P	1	1	1	0	0	0	0	50
Q	0	1	1	0	0	0	1	51
R	0	1	1	0	0	1	0	52
S	1	1	1	0	0	1	1	53
T	0	1	1	0	1	0	0	54
U	1	1	1	0	1	0	1	55
V	1	1	1	0	1	1	0	56
W	0	1	1	0	1	1	1	57
X	0	1	1	1	0	0	0	58
Y	1	1	1	1	0	0	1	59
Z	1	1	1	1	0	1	0	5A
								5B
								5C
								5D
^(FS)	0	1	1	1	1	1	0	5E
								5F

To convert to the ASCII value, ignore the P (parity bit) and add the hex value of B[6:0] to 20 hex.
For example for the / character, 0F + 20 = 2F

Tracks 2 & 3 Character Set - 4 Bit BCD with Odd Parity

Character	P	B_4	B_3	B_2	B_1	ASCII
0	1	0	0	0	0	30
1	0	0	0	0	1	31
2	0	0	0	1	0	32
3	1	0	0	1	1	33
4	0	0	1	0	0	34
5	1	0	1	0	1	35
6	1	0	1	1	0	36
7	0	0	1	1	1	37

Character	P	B_4	B_3	B_2	B_1	ASCII
8	0	1	0	0	0	38
9	1	1	0	0	1	39
:(AS)	1	1	0	1	0	3A
;(SS)	0	1	0	1	1	3B
<	1	1	1	0	0	3C
=(FS)	0	1	1	0	1	3D
>	0	1	1	1	0	3E
?(ES)	1	1	1	1	1	3F

To convert to the ASCII value, ignore the P (parity bit) and add the hex value of B[4:0] to 30 hex.
For example for the ; character, B + 30 = 3B

P = Parity Bit, usually odd
SS = Start Sentinel
ES = End Sentinel
FS = Field Separator
AS = Account Separator (Track 3 Only)
OG = Option Graphic

For more information logon to www.anadigm.com