Inside DISTO's 2-Meg Kit

Tony Distefano

Original OCR Version 10/04/2000

Inside the DISTO 2 Meg Kit

Introduction:

This brief will describe, in detail, how the DISTO 2 Meg Kit for the COCO 3 works. In order to fully comprehend the following brief, I will assume that you are already very familiar with CPUs, DRAM (Dynamic Random Access Memory), discrete logic gates, circuit diagrams and understand how the GIME chip of the COCO 3 works. It is also recommended that you have and understand the complete schematic diagram of the COCO 3.

Included is a full schematic diagram of both the DAT board and the Memory board as well as a full listing of the PAL (Programmable Array Logic) chip. The discussion will start with a short overview on what is required to get the COCO 3 to recognize and use 2 megabytes of memory. Then, a review of how the GIME works, followed by a theory of operation of DRAM refresh. Finally, there will be a 'chip by chip' circuit description and timings. Although I have spent a lot of time and effort into making this brief as accurate as possible, there is always a chance or errors.

Overview:

When I first thought about expanding the memory capabilities of the COCO 3, I knew it would require a lot of time trying to understand how the GIME treats memory. The 256K memory chips that the GIME uses has many working parameters. I wanted to increase its memory capabilities to 2 meg of memory. At the time, the most common memory upgrades were 1 meg SIMMs. I decided to use two of them to make my upgrade. I also wanted my upgrade to work with as many types of SIMM possible. So, I got as many kinds as I could find at the time. These were the 2 chip, 3 chip, 8 chip and 9 chip kinds. I had to make sure that my circuit worked with all of them. Whatever the case, there were a few electronic differences between the

---

chips on the SIMMs and the 256K memory chips currently used by the GIME. The first is the extra address lines. There are 2 more address lines on a 1 meg chip than there are on a 256K chip. The second is the refresh requirements are also more than the GIME can cope with. The final is not really a electronic difference as it is a logic difference. The GIME is not set up to handle the extra memory. The GIME uses a "bank switching" method to allow the CPU to access more than 64K, but the bank switching capability of the GIME does not go beyond 512K. A extension to the existing circuits has to be added.

That, along with a few more GIME quirks, is what's required to allow the COCO 3 to access and use 2 meg of memory. The following is everything I know about the GIME and my 2 meg memory upgrade kit.

The GIME:

The heart of the COCO 3 is a Motorola 681309E CPU. This CPU can directly access 64K of memory via 16 address lines. The COCO 3 also has ROM, RAM, I/O ports and a color video display circuit. Tying all these parts together is the GIME (Graphics/Interrupt/Memory/Enhancer) It is the job of the GIME to supply all timings and memory

mappings for the COCO 3. The GIME also includes the DAT (Dynamic Allocation Table) and video graphic display hardware.

The CPU has to operating speeds; 1 mhz or 2 mhz. The video circuits always run at the same speed of 2 mhz. Wait a minute, how can the CPU run at 2 mhz and video run at 2 mhz. Well, the GIME is always running at 4 mhz. It interleaves CPU and video at 1:1 when the CPU is running at 2 mhz. When the CPU is running at 1 mhz, the GIME skips every second CPU cycle and does nothing. It takes a rest. The GIME follows this sequence, CPU access then Video access then no access then Video access and then repeats the same operation. When the CPU is running at 2 mhz, the 'no access' is replaced with another CPU access. The CPU's clock system, E and Q clocks, change from 1 mhz to 2 mhz and back, but all memory access timings stay the same at 4 mhz or about 4 I/O cycles per micro second.

The GIME generates all system timing and does so with the help of a 28.63636 mhz crystal. The GIME uses this crystal and a multitude of counters and latches to generate video text and graphics, memory management, dynamic allocation, memory mapping, I/O mapping, ROM switching and many other functions. This chip is so complexed that it would require hundreds of logic gates to duplicate. The COCO 3 service manual describes the GIME in great detail. I will not attempt to duplicate it here but only review sections of it that pertain to the 2 Meg Kit.

The fact that the GIME is a complexed device and that it has many signals that are not available to the outside world via chip pins is what made the 2 Meg Kit so hard to design. If certain signals were available, then upgrading the COCO 3 to 2 meg of RAM would have been relatively simple. Instead, two boards with a bunch of chips and many hours of R & D was required.

DRAM Refresh:

A quick review on DRAM is necessary to understand refresh. First, this type of memory uses a multiplexed address line technique to save on the number of pins required to access great amounts of data. It uses two pins to strobe in two sets of address data. 256K memory chips require 18 address lines while 1M memory chips require 20 address lines. Dividing each address set into two gives 9 address lines to 256K chips and 10 address lines to 1M chips. The first signal is the RAS (Row Address Strobe) line. It is used to strobe in the first half the address lines. The second is the CAS (Column Address Strobe) line. It is used to strobe in the other half.

To save space on the chip surface and production cost, the memory cells of DRAM are made up of tiny capacitors. When the capacitor is charged above a fixed internal value, it is considered logic 1 or HI. When the capacitor is below this level it is logic 0 or LO. The shortfall of this technique is that the voltage across the capacitor slowly decays and drops down to below the internal threshold all by itself. To avoid this, the capacitor must be topped up every so often.

This is 'Refresh'. Most DRAM chips have several methods of refresh available. The easiest refresh mode is a simple read cycle. The GIME uses this method of refresh. The COCO 3 with 512K memory uses 256K DRAMs. These chips require a 256 cycle refresh and must be repeated every 4mS. The GIME accomplishes this by doing 10 video read cycles during every Hsync (horizontal sync). Hsync happens every 63uS therefore 256 cycles takes just under 2mS to complete. This is well under the required 4mS for this type of DRAM.

Chip by Chip:

When reading this section, have the schematic diagrams beside you for quick and easy reference. I refer to them continuously. Whenever I mention more than one signal and/or pin number together, they are always respectively related. The DISTO 2-Meg Kit is divided into two sections, the DAT board and the Memory board. Let's start with the DAT board description. J1 gives the DAT board access to all the pins of the CPU. This is required in order to have access to all of the address, data and control pins of the CPU. J2 is the junction between the DAT board and the

Memory board. There are only 4 signals required between the two boards. Two of them (pin #2 and #4) are the last two bits (D6 & D7) of the DAT memory. Basicly these two bits act like a 512K bank switcher. With two bits, you can control 4 banks and 4 banks of 512K each equals 2 megs of memory. The third (pin #3) is the E clock, required by the Memory board for proper timing and the forth (pin #1) is the CPU/*VID output generated by the Memory board. There are two users of memory. The CPU is one and the video circuit is the other. This signal tells the bank switcher circuit whether the current cycle is a CPU cycle or a video cycle. More on that signal later.

U6 is a PAL chip and is used strictly for memory mapping. It otherwise would have taken several standard logic chips to achieve the same results. The PAL used is a generic 18CV8. All the signals to and from this chip are labelled according to function. U5 is also used for memory mapping. It selects the higher order of address lines into the PAL. With that, the PAL is able to address single locations of memory without mirroring. There are only four outputs from the PAL. The

---

first, 01, is pin #19. It is a write only memory select from $FFA0 to $FFAF. The second, 02, is pin #18. It is a write only memory select for only $FF91. The third, 03, is pin #17. It is a write only memory select for only $FF9B. The forth, 04, is pin #16. It is a read/write select for $FE00 to $FEFF. The remainder of the pins are taken up by the various address and control lines needed to properly decode these signals. A full JEPEC listing of this PAL is listed at the end of this brief.

U1 through U4 form the DAT memory D6 and D7. These 74LS670s are dual access memory chips. Each can access 4 bits in 4 locations. Since the DAT board only needs 2 bits, only two are used. They have separate input/output as well as separate address select pins. These chips have two separate modes of operation. The DAT Read/Write mode and the Memory Write mode.

In the Memory Write mode, the CPU can write to these chips as if they were, 'write only' memory mapped. They are memory mapped to the same locations as the GIME's DAT memory, from $FFA0 to $FFAF. The data lines are connected to the missing D6 and D7. The DAT diagram labels which are which. Since the required memory is 16 address locations, 4 chips were needed. WA and WB of all 4 chips are connected to A0 and A1 of the CPU. U7A is a 2-TO-4 decoder used to decode A2 and A3 into 1 of 4 chip selects (GW). The enable pin of U7A (G) gets is memory map from U6.

In the DAT Read/Write mode, these chips function in the same way as the DAT memory in the GIME but only to D6 and D7 which do not exist on the GIME. RA and RB of all 4 chips are connected to A13 and A14 of the CPU. These along with A15 act as the DAT decoder. A15 and the output of U9A form the input to U7B which is a 2-TO4 decoder. (The output of U9A duplicates the TASK register in the GIME. This duplication is necessary because, the output of the TASK register needed to reflect which TASK group is active, is not available outside the GIME. The TASK register is D0 of address $FF91 and is write only in this circuit.) This selects which of the 4 chips is needed.

In any video text or graphics mode, the GIME can access a

---

maximum of 512K of memory. The GIME can place video anywhere within thus 512K area. The problem with 2 megs of memory is that the GIME does not know how to deal with the extra memory. Built onto the DAT board is a circuit that divides the 2 megs of memory into four 512K banks. U9B and U l 0A are 2 write only, latched bits, D0 and D 1, memory mapped to location $FF9B. These two bits control these 4 banks of video memory. On power/up or reset, these two bits are 00. This lets the GIME access the first 512K bank of memory during the video cycle. Writing a number from 0 to 3 into this area will select which bank of video memory the GIME will use. This allows the user to switch banks at any time, in order to access any part of the full 2 megs of memory for video. There is however, one limitation.

The GIME cannot switch banks on its own. This means that the user must make sure that any video text or graphics screen does not cross a 512K boundary.

The last chip on the DAT board is U8. This chip (74F153) is a dual 4-TO-1 encoder. This chip takes in the two DAT bits (D6 & D7, which basicly represents four 512K banks) and the two video bank bits and encodes them into one of four modes. The four modes are controlled by inputs A & B of U8. The B input comes from the Memory board. Circuits over on the Memory board generate a signal called CPU/*VID. When this signal is HI (logic 1), the GIME is generating a CPU memory cycle. When it is LO (logic 0), the GIME is generating a VIDEO memory cycle. This signal is needed to select between the complex data of the DAT memory and the simple two bit data of the video bank bits. The A input of U8 roots from the PAL. This signal is active when the CPU accesses a read or write in the area from $FE00 to $FEFF or simply $FEXX. The value for X is 'don't care' or 'any number'. When active, LO (Logic 0), this means that the CPU is accessing this $FEXX area. There is a special condition when the CPU access this area, it must always be the same area, no mater how the DAT memory is set. When there was only 512K in the COCO 3, this was no problem, the GIME would make sure that this was so. But, with the addition of three more banks of 512K (4 banks total), a special condition had to exist to make sure that whenever this area was accessed, it would be the same. Bank 0. This A input insures this. However, thus condition has to be true only when the CPU wanted that area. The video circuit is exempt of this condition. That is the forth

---

mode. The following is a description of the 4 modes of U8.

Mode 0: A=0, B=0

This is a video cycle while the CPU is accessing $FEXX. In reality, this mode can never happen, but rather than having the inputs floating, they are tied to the Video bank bits.

Mode 1: A=1, B=0

This is a video cycle while the CPU is not accessing $FEXX. Now this is the real video mode. Not only is the CPU not accessing $FEXX but it is not accessing anything. The outputs of U8 are the data from the Video bank bits. In this mode, the GIME is accessing video data from one of the four 512K banks set by the two Video bank bits.

Mode 2: A=0, B=1

This is a CPU cycle while the CPU is accessing $FEXX. In this mode the two output bits are forced to LO. This selects bank 0. The data in the DAT memory is being ignored and bank 0 is being selected. The data inside the DAT memory of the GIME is also being ignored and forced to 0, therefore always accessing the same 256 bytes of real RAM. Circuits inside the GIME are taking care of this. All this circuit has to do is present bank 0 during this mode.

Mode 3: A=1, B=1

This is a CPU cycle while the CPU is accessing memory other than $FEXX. This is the mode when the data in the DAT memory switches to one of the four 512K banks. With the DAT memory inside the GIME and the DAT memory held in this adapter, the CPU can access anywhere in 2 meg address space.

That is just about it for the DAT board. Now we move over to the Memory board. This board has three duties. The first duty is to host the memory SIMMs. U101 and U102 are standard SIMM sockets. They each hold one 1-Meg SIMM. Now, I have tried just about every kind of SIMM I could get a hold of in this board. The 8-Chip, 9-Chip, 2Chip and 3-Chip all work with this board. I have tried from 120ns to 80ns speeds and they also worked. So, just about anything that is called a ' 1-Meg SIMM' works. There are three connectors on the Memory

---

board. J1'101 and J1'102 connect to the COCO 3's regular mem Tory expansion connector. JP103 is the connector that shuttles over to the DAT board and to two required signals on the COCO 3's motherboard.

The second duty the memory board is to multiplex the two DAT bits into A9 (explained earlier). This is done with one half of U104. U104 is a dual 4-TO-1 encoder, functionally identical to U8 of the DAT board. On the Memory board schematic you can see that DAT6 is connected to both 2C0 and 2C2. Similarly, DAT7 is connected to both 2C1 and 20. This limits this half of the encoder to only a 2-TO-1 encoder. Only the A input will switch signals. The logic level of the B input will not change the output. (I did this strictly as a trick to economize chips. I used the other half of this chip to do another function using only the B input. Otherwise, two chips would be needed.) The A input is used as a RAS/CAS selector. With proper timing from U103, when A is LO, DAT6 will become the RAS address of A9. When A is Hi, DAT7 will become the CAS address of A9. In other-words, DAT6 and DAT7 become the multiplexed address A9 of the 1-meg SIMMs.

The other half of U104 is used to alter the CAS signal to force the SIMM into an 'internal refresh cycle' mode. The mode used is the CAS-Before-RAS Refresh. In this half, CAS from the COCO 3's GIME is routed to 1C2 and 1C3. 1C0 and 1C2 are grounded. This configuration also limits this half of the encoder to a 2-TO-1 encoder. The A input of this chip will have no effect on the output. Only the B input will select ground (logic LO) or CAS to the output 1Y. I will discuss the source of the B input in detail a little later. When the B input is HI, the output, LOUT, (which is connected to the CAS input of the SIMMs) will follow the CAS signal. When the B input is LO, the output, COUT, will also be forced LO.

We now know how U104 is used to control CAS and multiplex A9. The last four chips are used to generate the proper timing to control these go signals. These four chips represents the lion's share of time and effort to design. It is the heart of the 2-Meg adapter. In order to control CAS and to multiplex A9 at the proper time, all timing sequences had to be in sync, somehow, to the GIME. It would have been easier to design the 2-Meg adapter if certain signals were available

---

from within the GIME, but they arc not, therefor they have to be synthesized. Three timing signals are needed. The first signal needed is to know when the CPU is accessing memory and Video is accessing memory. The second is when to switch between the RAS and the CAS address. The third is knowing, when and how, the GIME is doing refresh. To find out how something works without having the benefit of original diagrams and theories is called 'reverse engineering'.

After a lot of time spent with a logic analyzer, I found a common signal. All signals had a constant relation with the E clock. With the knowledge that everything is in sync with the falling edge of the E clock, this was a good point to start. Since the 28 mhz clock is the shortest time in which anything can happen, any timing edge or window can be derived. U 103 is a loadable synchronous up/down counter. The clock to U103 comes from a buffered (Ul06B) 28 mhz. U105A is used to load a preset value into the counter. The output Q of U105A drives the Load input of the counter. Because the PR (preset) is connected to 28 mhz, Q is continuously being preset to its HI state. The CLK input to U 105A comes from an inverted E clock via U 106A because this gate uses the rising edge. Therefore, the falling edge of the E clock (inverted to the proper edge for this gate) will transfer D (which is connected to GND or logic LO) to Q. This will cause Q to go LO. But not for long, because the PR is being driven by the 28 mhz. Q will only stay LO until the next LO cycle of the 28 mhz which is only about 18 nano seconds later, which by the way is just about the speed limit of this counter. In other words, the falling edge of the E clock resets the counting operation of U103.

Now, what we have is a 4-bit counter in sync with the falling edge of the E clock. This 4-bit counter gives 16 discrete positions and 32 edges in which to choose the signals needed. Nothing, in the COCO 3, happens faster than these positions and edges. Knowing that, the next step is to find out where in these 16 positions our signals are placed. We need just two signals. The first is a signal that will present the RAS address and the CAS address in proper timing and sync with the RAS and CAS of the GIME. (This signal is not known because it is not available outside the GIME.) This was done by looking at the 4 outputs of U103 and the RAS/CAS signals on a high speed oscilloscope. By setting up to scope to see the time between RAS going LO and CAS

going LO on one channel and looking at each output one at a time on another, I was able to see which output fell exactly between them. QB of U103 turns out to be the perfect signal. It allowed the RAS/CAS address to be stable and switch in the precise time that the GIME uses to control the other 9 address lines. More reverse engineering!

The other signal needed in my circuit is the CPUVID selector. This signal has two uses. The first is to allow CPU access to DRAM during its Read/Write cycle and allows the VIDEO circuit access to DRAM during its Read cycle. With more scope watching of the E clock and the CAS signals, QD of U103 has exactly that needed output. It is HI during any CPU access and LO during any VIDEO access. This output is fed back to the DAT board to select the proper data via the B input of U8.

The second use of CPUVID is to trigger a refresh cycle for DRAM. To properly refresh DRAM is the third duty of the Memory board. As previously explained, the only option available to refresh DRAM without using many chips as counters and selectors is the CAS before RAS refresh. To use the internal DRAM refresh counters in this mode, all this circuit has to do is to force the RAS/CAS lines into a CAS before RAS refresh cycle. Given that the GIME uses video cycles during the H-Sync period to do its refresh (many hours of reverse engineering), this circuit uses the same period. Given that during the H-Sync period, video is used to do refresh, this circuit uses these video cycles and converts them into CAS before RAS refresh cycles. The easiest way to generate this type of cycle is to extend the time normally given to a 'CPU only' cycle and create a 'double long' cycle. By creating this 'double long' 'CPU only' cycle, the video cycle is lost. No big loss there, they were used to do refresh anyway. Instead of the GIME doing a 'read only' refresh cycle, a CAS before RAS refresh cycle is generated. All this circuit does is to switch from a Read cycle refresh (using the GIME's internal counters) to a CAS before RAS refresh (using the DRAM's internal counters). This method allows the GIME to use the longer '512 cycle refresh' requirements of 1Meg DRAM without having to extend the GIME's internal counters.

After more scope gazing, I found that the rising edge of CPUVID was a good place to start. It is unaffected by the dual speed

mode. This edge is too early for CPU CAS, but the falling edge is just a little too tight for comfort. So, a short delay is needed. The rising edge of CPUVID drives the B input of U107A. On this edge, Q goes high until about the middle of the CPU CAS. The output of Q goes into the A input of U107B. The falling edge of Q (U107A) causes *Q of U107B to go LO. The length of *Q is adjusted so that it stays LO until the middle of the VID CAS. To sum up, *Q, pin #12 of U107B, goes LO from the middle of the CPU CAS to the middle of the VID CAS. This output drives the B input of U104. Now, this half of U104 is configured as a two to one multiplexer, because of the way it is wired. When B is HI, CAS (pin #3 & #4) is output to COUT. When B is LO, GND is output therefor forcing CAS to stay LO during the time between the CPU CAS and the VID CAS. This action forces the DRAM chips to do an internal refresh cycle.

Now this type of cycle cannot go on for ever, there would be no video otherwise. There is a way to control this. Since the GIME does refresh only during Hsync, I gated U107A to output a pulse only during Hsync. The A input of U107A is from Hsync of the GIME. When A is III, no pulses are generated, therefore no double long CAS, therefore normal operation of CAS. When A is LO, during Hsync, pulses are generated, therefore double long (refresh) cycles are in effect.

Conclusion:

Well, that's about it. When you put all of these circuits together in the COCO 3, 'it can access 2 mega bytes of memory.

JEPEC PAL Listing

```
Name COCO2MEG;
Partno DIS00020;
Date 02/10/93;
Revision 02;
Designer Tony DiStefano;
Company DISTO;
Assembly DAT Board;
Location U6;
/************************************************/
/* */
/* COCO2MEG memory map decode logic */
/* */
/************************************************/
/* Allowable Target Device Types: PAL16L8 EP320 */
/************************************************/
/** Inputs **/
Pin [1..9] = [A0..8] ; /* CPU Address Lines */
Pin 11 = E ; /* CPU E CLOCK */
Pin 13 = !WR ; /* CPU Write when LO */
Pin 14 = Q ; /* CPU 0 CLOCK */
Pin 15 = !AX ; /* 1111,111X,XXXX,XXXX = LO */
Pin 12 = CPU ; /* CPU = HI VIDEO = LO */
/** Outputs **/
Pin 19 = !FFAX ; /* Write Data to REGISTERS */
Pin 18 = !FF91 ; /* Task Register Bit */
Pin 17 = !FF9B ; /* Video Bank Bit */
Pin 16 = !FEXXM ; /* Disable REG when $FEXX */
/** Declarations and Intermediate Variable Definitions **/
/** Logic Equations **/
FFAX = Q & E & WR & AX & A8 & A7 & !A6 & A5 & !A4;
FF91 = Q & E & WR & AX & A8 & A7 & !A6 & !A5 & A4
& !A3 & !A2 & !A1 & A0;
FF9B = Q & E & WR & AX & A8 & A7 & !A6 & !A5 & A4
& A3 & !A2 & A1 & A0;
```

```
FEXXM = AX & !A8 & CPU;
FFAXW = E & AX & A8 & A7 & !A6 & A5 & !A4;
```