# Interfacing the CY7C611A with the VIC64

The popularity of the VMEbus and the Motorola 680x0 family of microprocessors has produced a large number of peripheral controllers with 680x0-compatible asynchronous local bus interfaces. Many of these parts are mature, proven, and inexpensive, making them attractive candidates for low-bandwidth I/O applications.

This application note describes an interface between the synchronous CY7C611A SPARC processor and asynchronous bus peripherals such as the Cypress Semiconductor VIC64 64-bit VMEbus interface chip. It is based on the design of a SPARC-based VIC64 VMEbus evaluation board developed by Cypress Semiconductor. Only the synchronous-to-asynchronous bus conversion logic is discussed within this application note; however, the full schematics of the board and all PLD design files are available from Cypress Semiconductor.

## Related Documents

The reader may also wish to consult the following documents for additional information:

- *VIC068A/VAC068A User's Guide*
- *VIC64 and CY7C964 Design Notes*
- "Memory Protection and Address Exception Logic for the CY7C611A SPARC Controller" application note
- "Understanding the 361" application note
- Motorola's *MC6800 Family Reference*

With the exception of the Motorola document, these documents are available through your local Cypress Semiconductor field sales office.

## Typical Asynchronous Bus Operation

Asynchronous buses operate using some type of handshake system. The processor presents or requests data from a peripheral and an acknowledge is generated by the selected device. The length of the processor cycle is determined by the performance level of the peripheral. The processor maintains a bus cycle until it receives an acknowledge.

With this type of bus, operation problems can occur if the processor attempts a cycle to an address region that does not select valid memory or peripherals. In this situation an acknowledge signal is not issued to the processor and the system operation halts.

To avoid this potential lock-up condition, most asynchronous bus protocols have a separate signal for acknowledging erroneous cycles. Assertion of this signal releases the processor from the pending bus cycle and can also be used to inform the system software that the bus cycle did not terminate properly. These cycles are typically known as bus error and memory exception cycles.

## Memory Exception Cycles are Important

Asynchronous microprocessor buses are not unique in the inclusion of memory exception cycles. The CY7C601A and CY7C611A include a similar mechanism. In normal system operation, memory exceptions should not occur regularly. They can be used to furnish beneficial debug and system configuration information in some applications.

VMEbus applications where logic boards can be added and removed from systems often use the bus error mechanism to determine system configura-

tion. CPU board initialization software can *hunt* the VMEbus address regions, searching for other cards. Address regions that respond with normal acknowledge signals can then be further interrogated and initialized.

## Overview of the CY7C611A Memory Interface

The CY7C611A is a 32-bit, four-stage, pipelined SPARC RISC integer processor. The processor is synchronous and, after initializing the pipeline, it can execute one instruction per clock cycle.

The CY7C611A memory interface consists of a group of signals that control memory loads/stores, pipeline control, and memory exception generation. These signals are listed in *Table 1*.

**$\overline{\text{MHOLD(A/B)}}$**

These two signals are logically ORed together within the CY7C611A. Asserting either of these signals (Low) freezes the processor's pipeline, causing the processor to remain on the same execution cycle. The $\overline{\text{MHOLDA}}$ and $\overline{\text{MHOLDB}}$ signals allow the processor to communicate with slow peripherals.

**$\overline{\text{MDS}}$**

MDS is used to strobe data or instructions into the processor after the pipeline has been frozen by the assertion of $\overline{\text{MHOLD(A/B)}}$. Asserting $\overline{\text{MDS}}$ with the pipeline frozen enables the processor to clock the information present on the external data bus into the processor. MDS is also used to strobe in the $\overline{\text{MEXC}}$ signal.

**$\overline{\text{MEXC}}$**

Asserting this signal (Low) informs the processor that the memory system could not supply the data or instruction requested. When the signal is asserted, either a data or instruction access trap occurs. The type of trap directly corresponds to the type of memory cycle in progress. $\overline{\text{MEXC}}$ is strobed into the processor by asserting the $\overline{\text{MDS}}$ signal.

**Table 1. CY7C611A Memory Interface Signals**

| Name | Description | Type |
|---|---|---|
| $\overline{\text{MHOLD(A/B)}}$ | Memory Hold A/B | Input |
| $\overline{\text{MDS}}$ | Memory Data Strobe | Input |
| $\overline{\text{MEXC}}$ | Memory Exception | Input |
| $\overline{\text{INULL}}$ | Integer Unit Nullify | Output |
| $\overline{\text{WE}}$ | Write Enable | Output |
| $\overline{\text{WRT}}$ | Advanced Write | Output |
| $\overline{\text{RD}}$ | Read Access | Output |

**INULL**

The assertion of INULL (High) indicates that the memory cycle in progress is being nullified. Memory cycles are nullified when the processor determines the the current address is invalid or that the information being read is not required. This improves performance because no time is wasted communicating with slow peripherals or reloading cache line data that is not needed. INULL is asserted by the processor in the following situations:

- During the second cycle of any store operation. The same address is presented on the first and second cycle of all store operations, the second occurrence is nullified because it is not truly the *next* address being requested by the processor.
- On all traps. This nullifies the third instruction fetch after the trap is encountered, because the processor vectors to the appropriate trap handler.
- On a load with the hardware interlock active.
- On JMPL and RETT instructions.

**$\overline{\text{WE}}$**

Write Enable (active Low) indicates that the processor is performing a store operation. This signal is asserted in the second clock cycle of the store operation, the same cycle that the store data is presented.

**WRT**

Advanced Write (active High) notifies the external control logic that a store operation is in progress. The processor asserts this signal on the first cycle of the operation, before the data is available.

**RD**

Read Access (active High) indicates that a load cycle is in progress.

## CY7C611A Load and Store Cycles

Two general bus cycles, load and store, are described at a high level of abstraction within this section. Many variations of these cycles exist.

When loading data, the processor supplies the address information on the rising edge of a the processor clock and expects the data on the next rising edge. The Read signal (RD) remains active (High) during the cycle with $\overline{WE}$ and WRT inactive (High and Low respectively).

The process for storing data is similar, but one additional clock cycle occurs before the processor presents the data. On the first cycle of the store the address is presented, RD is driven inactive (Low), and WRT is driven active (High). On the second clock cycle, the store address is again placed on the bus, WE is asserted (Low), WRT is deasserted (Low), and the data is placed on the bus. INULL becomes active after the falling edge of the second clock cycle, nullifying the second occurrence of the store address.

*Figures 1* and *2* show Store Single and Load Single CY7C611A bus cycles. In general, the address execution unit operates one clock cycle ahead of the data unit. The cycles shown assume that the peripherals or memory are capable of operating at the performance level of the processor. The pipeline is never frozen using MHOLDA or MHOLDB, and both cycles terminate normally without generating memory exceptions.

## Overview of the VIC64 Asynchronous Interface

The Cypress Semiconductor VIC64 is compatible with the 680x0 asynchronous microprocessor bus. It is a 64-bit VME interface chip capable of performing D16, D32, and D64 block transfers on the VMEbus at transfer rates up to 70 Mbytes/sec. The VIC64 and its associated control logic are also capable of performing Direct Memory Access (DMA) operations during VMEbus block transfers. VIC64 DMA operations generate 680x0-compatible bus cycles to transfer data to and from local memory.

The basic control signals required to communicate with a VIC64 or other generic 680x0-style peripherals are listed in *Table 2*.

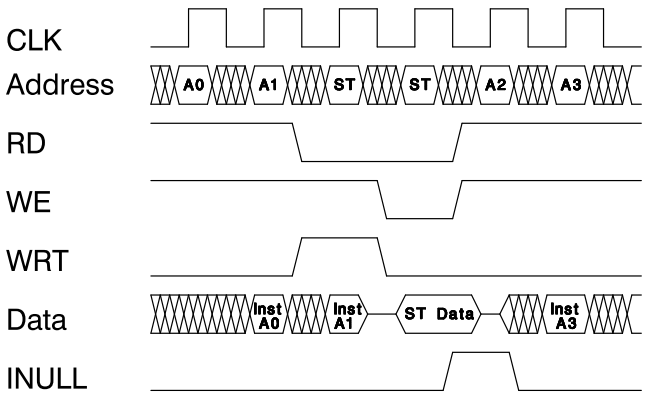



**Figure 1. CY7C611A Store Single Operation**

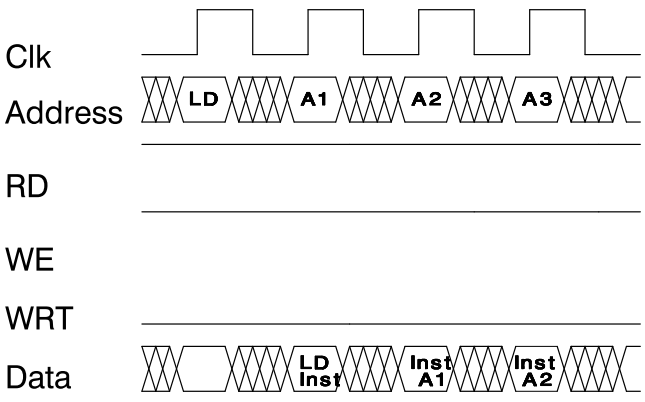



**Figure 2. CY7C611A Load Single Operation**

**Table 2. 680x0 Basic Control Signals**

| Name | Description | Type |
|---|---|---|
| AS | Address Strobe | (Normally) Input |
| DS | Data Strobe | (Normally) Input |
| R/W | Read Write | (Normally) Input |
| DSACK0/1 | Data Acknowledge | (Normally) Output |
| BERR | Bus Error | (Normally) Output |

The signal types, input or output, have been referenced in a normal operating mode for dumb peripherals. Since the VIC64 is also capable of becoming a *bus master* during local DMA transfers, it can source AS, DS, and R/W as well as receive these signals. This also holds for the output signals DSACK1/0 and BERR. If the VIC64 is generating the bus cycle, these control signals become inputs.

**AS**

Address Strobe is asserted (Low) at the beginning of a bus cycle to indicate that a valid address is currently on the address bus. The address must remain constant while Address Strobe is active. Address Strobe remains active for the length of the bus cycle. On the VIC64 this signal is named Processor Address Strobe (PAS).

**DS**

The assertion of Data Strobe informs the receiving peripheral device or memory that it may place data on or extract data from the bus.

**R/W**

The Read/Write signal indicates the type of cycle in progress. This signal is High for read cycles and Low for write cycles.

**DSACK0/1**

The DSACK0/1 signals are driven by the peripheral device to tell the device performing the bus cycle that the data has been accepted or is available on the bus. Bus cycles persist until an acknowledge or BERR signal is detected. There is no limit to the length of this type of bus cycle. Many 680x0 peripheral devices have only a single acknowledge, often named *DTACK*.

VIC64 has two DSACK signals, 0 and 1, which adhere to the Motorola dynamic bus sizing convention and report the bus width, (8, 16, or 32 bits), of the peripheral acknowledging the bus cycle.

**BERR**

Asserting BERR terminates a pending bus cycle and forces the processor to trap to an exception handler. This signal terminates erroneous bus cycles. Many systems have bus timeout timers that monitor the length of all bus cycles and assert BERR if a cycle persists for the timeout period.

## 680x0 Asynchronous Read and Write Cycles

As with the corresponding section on the CY7C611A load and store cycles, the read and write cycles within this section are only described at the High level. Many variations of these cycles exists. Refer to the *VIC068A/VAC068A User's Guide* or the Motorola microprocessor documentation for more information.

Write cycles begin with an address being placed on the bus by the controlling processor or peripheral. The R/W signal is driven Low to indicate a write cycle. Address Strobe is asserted (Low), denoting the beginning of the cycle. One clock later, after the data has been placed on the bus, DS is asserted (Low). All signals remain stable in this state until a normal acknowledge, DSACK0/1, or error acknowledge is received (Low).

Reads cycles operate in a similar manner. An address is placed on the bus by the controlling peripheral and R/W is driven High to indicate a read cycle. AS and DS are driven Low simultaneously, informing the peripheral that data can be placed on the bus. These signals remain in this state until an acknowledge of some sort is received.

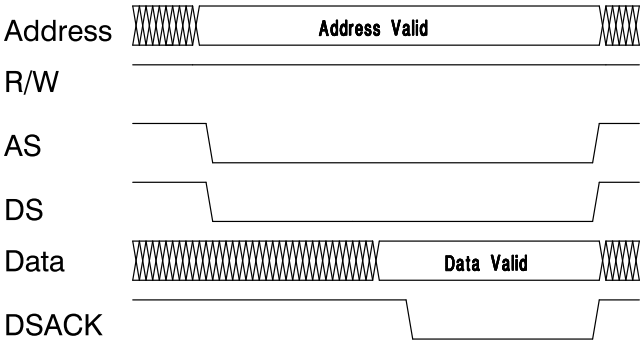



**Figure 3. 680x0-Compatible Read Cycle**

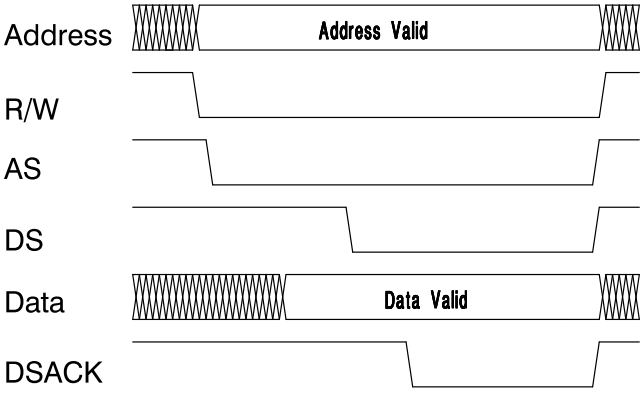



**Figure 4. 680x0-Compatible Write Cycle**

*Figures 3* and *4* show typical bus cycles for asynchronous 680x0 peripheral devices like the VIC64.

## Clear Differences in Cycle Types

As can be seen with even a cursory view of the two styles of bus cycles, interfacing between the CY7C611A and peripherals like the VIC64 can be challenging. In general, the problem is slowing the CY7C611A down to operate with the peripheral. This can be accomplished in a number of ways, each having its own set of considerations.

## Pipeline Freezing Using MHOLDA/B

Per design, the CY7C611A contains control logic that allows the execution unit to be held for communication with slow memory devices or peripherals. The logic sequences required to suspend execution have some tight timing requirements. If an MHOLD signal is not asserted quickly enough, the processor advances to the next cycle. The CY7C611A, unlike the CY7C601, does not have an MAO pin. Therefore, if the processor does advance to the next cycle, there is no way to have it place the last address back on the bus. This can become a significant problem. Obviously other undesirable situations can occur when control logic does not or cannot meet necessary timing constraints.

These potential problems can be overcome by using high-performance logic like the CY7B336, CY7B337, CY7B338, and CY7B339 family.

## Clock Stretching

Another method of interfacing the CY7C611A to slow memory and peripherals is a procedure known as clock stretching. The CY7C611A is a fully static microprocessor. This furnishes a simple method for slowing the processor down, simply by delaying or changing the duty cycle of the clock. The processor can be held within an execution state without asserting $\overline{\text{MHOLDA/B}}$. This technique allows execution to resume without strobing data into the processor with $\overline{\text{MDS}}$.

This procedure works well for peripherals with fixed access times. When the bus cycle begins, the clock is stretched. When the peripheral has completed the data transaction the clock is allowed to advance.

There are two subtle problems with this method of interfacing:

- Additional logic is required to operate with peripherals that are truly asynchronous in nature
- Memory exceptions cannot be generated because they require $\overline{\text{MDS}}$, $\overline{\text{MEXC}}$, and $\overline{\text{MHOLD}}$

Each of these problems becomes a significant issue when interfacing to the VIC64. Using the VIC64 to perform single-cycle processor transfers across the VMEbus has no guaranteed cycle time. The length of the cycle is directly dependant on the performance level of the slave plus the acquisition time required to obtain the VMEbus. Therefore, using a fixed clock-stretch cycle time would either be too short for slow slave boards, or a significant performance barrier when communicating with faster boards.

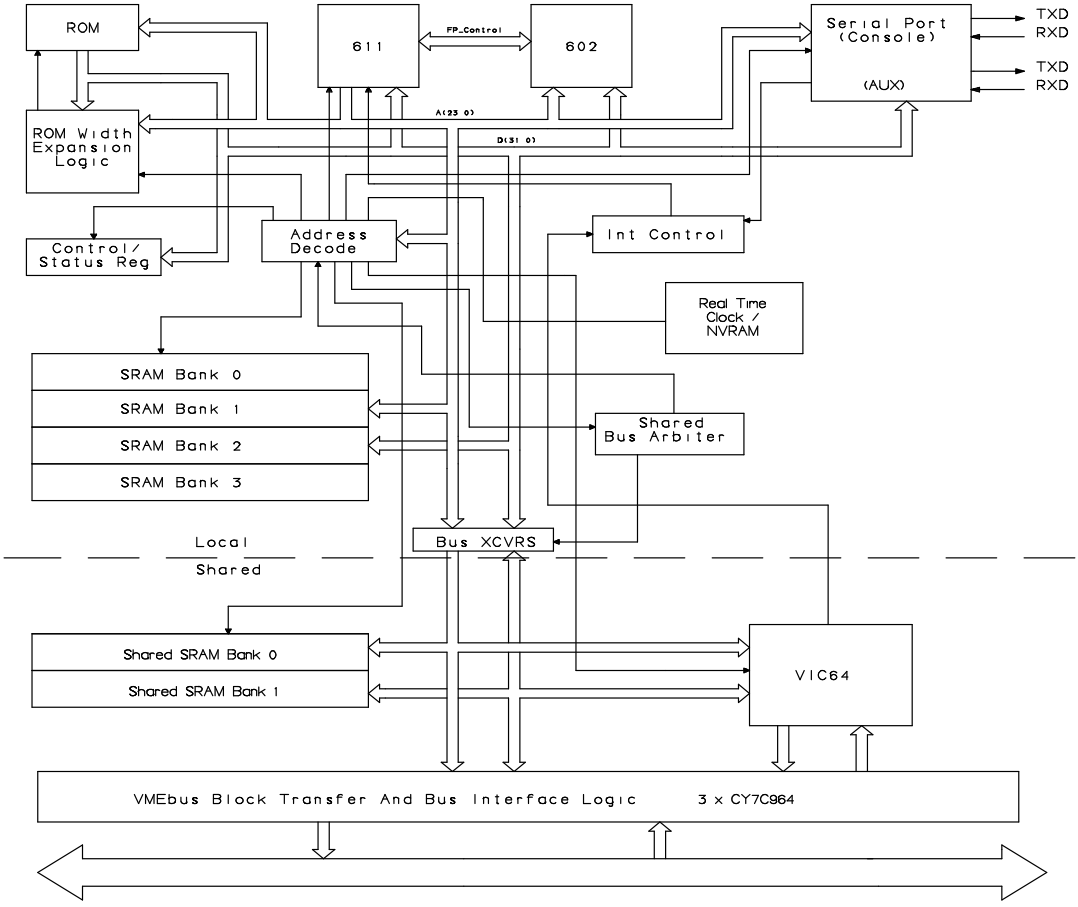



**Figure 5. CY7C611A / VIC64 VMEbus Board Block Diagram**

Bus errors are also an integral part of the VMEbus and the VIC64's operation. The inability to use this feature would significantly limit the functionality of many systems. Mapping this function into an interrupt is not desirable because if interrupts are disabled, or if interrupt latency is encountered because higher-priority interrupts are pending, the software's ability to determine the cycle that caused the error is hampered.

## CY7C611A/VIC64 VMEbus Board

The CY7C611A/VIC64 evaluation board is a typical single-board computer with the following features:

- 25-MHz CY7C611A embedded-control SPARC RISC processor
- 25-MHz CY7C602 floating-point unit
- 64 Kbytes to 4 Mbytes of local SRAM
- 64 Kbytes to 2 Mbytes of dual-port SRAM
- 128 Kbytes to 512 Kbytes of EPROM
- MC68681 DUART
- 2 Kbytes of non-volatile storage
- Time-of-day clock calendar
- Split address and data bus for high-performance VMEbus block transfer operation

The block diagram of the board is shown in *Figure 5*. The local SRAM on the board operates at zero wait states, removing the need for an instruction or data cache. With the exception of the local SRAM and a

system control/status register, all other peripherals operate using asynchronous 680x0-style bus cycles.

Having all peripherals operate using one of the two cycle types simplifies the interface and control logic. The 680x0-style cycle is essential since the VIC64 and MC68681 DUART communicate on this type of interface. The shared SRAM also needs to operate using this type of cycle to be compatible with the VIC64 during VMEbus block transfer DMA operations. It is then simple to adapt other slow peripherals (ROM, non-volatile SRAM, and Time-of-Day clock) to the slow, 680x0-style bus cycle.

## The CY7C611A-to-680x0 Bus Converter

As discussed in the previous sections, there was a strong desire to build an interface that was logically simple but preserved memory exception capability. The scheme was implemented as a hybrid technique using the CY7C611A pipeline freezing and memory exception logic along with a clock-stretching technique.

The control logic is implemented within two PLDs, a CY7C361 and a 22V10B, operating as pseudo master slave devices. The logic is split between two devices because of other functionality needed on the board, which is well suited for the CY7C361. If these other functions were removed from the CY7C361, the entire synchronous to asynchronous conversion logic could fit within the CY7C361. However, the CY7C361 on the CY7C611A/VIC64 board provides:

- Generation and control of clocks for the processor and peripherals
- Local bus arbitration for the VIC64 and CY7C611A
- Synchronization of asynchronous signals, which is needed for the slave 22V10B

This bus conversion scheme operates as follows. The processor begins execution and an address is presented, latched, and decoded. If the address region decodes to a slow 680x0-compatible cycle, the clock to the processor and control logic is stretched. If the cycle terminates normally, the clock is re-enabled to the processor and to control logic, which advances to the next execution cycle. If the cycle terminates in a memory exception or bus error, $\overline{\text{MHOLDA}}$ is asserted to the processor, freezing the pipeline, and the clock is re-enabled. With the pipeline frozen and the processor and control logic clock running, $\overline{\text{MEXC}}$ and $\overline{\text{MDS}}$ are asserted to the processor, generating the exception.

## Clock Control Using The CY7C361

To simplify interface design and maximize performance, microprocessor control logic typically needs to operate at twice the clock frequency of the processor. Even with the relatively slow 25-MHz clock frequency of the CY7C611A, routing, managing skew, and operating TTL control logic at 50 MHz can significantly increase the complexity of a design.

To eliminate this problem, the CY7C361 was selected as a clock-generation device. The CY7C361 is an ultra high speed PLD that features an internal clock doubler, double input registers for metastable hardening of asynchronous inputs, and 32 general-purpose state macrocells. While this is not a typical application for a PLD, the CY7C361 has a pin-to-pin skew of 2 ns maximum.

Operating the CY7C361 at 50 MHz externally and 100 MHz internally allows the generation of three different 25-MHz clocks. While the system still requires a 50-MHz clock, the CY7C361 is the only device operating from it, simplifying routing and termination problems. Since no other device on the board operates from the 50-MHz clock, no relationship needs to be maintained between the CY7C361 clock input and output pins, removing the clock-to-output propagation delay from the timing analysis. The 25-MHz clocks operate all sequential logic on the board with the exception of the 3.68-MHz clock needed by the MC68681 DUART for baud-rate generation.

## The Clock-Generation Machine

The clock-generation state machine within the CY7C361 has the following input and output signals:

**NNULL (Input)**

This synchronous input is a conditioned active-Low signal formed by combining the CY7C611A INULL and the CY7C602A FNULL signals. FNULL is the corresponding nullify signal from the floating-point unit. It operates in the same manner as INULL. The NNULL signal is used to filter out the nullifies that occur during every store cycle. The store nullifies were a don't care for the board's control logic since the signal is generated to nullify the second occurrence of the store address.

$$\overline{\text{NNULL}} = (\text{INULL OR FNULL}) \text{ AND LWE}$$

INULL and FNULL are active High and LWE is simply the latched WE signal from the CY7C611A. LWE is latched on the rising edge of CPUHCLK.

**LWRT (Input)**

This synchronous input is the latched WRT signal from the CY7C611A. This signal is latched on the rising edge of CPUHCLK.

**$\overline{\text{MHOLDA}}$ (Input)**

This is the synchronous CY7C611A $\overline{\text{MHOLDA}}$ signal. This signal is generated by the 22V10B that generates Motorola-style bus cycles.

**DONE (Input)**

A synchronous signal generated elsewhere within the CY7C361 that indicates that a Motorola bus cycle has been acknowledged. All acknowledges returned from the board are asynchronous signals. Double input registers on the CY7C361 are used to synchronize it for state logic use. DONE is active Low and is asserted if the cycle terminates normally or in a bus error.

**HOLD (Input)**

HOLD is a synchronous signal from the address decoding logic indicating that the selected peripheral requires a slow, 680x0-style bus cycle and that the CY7C611A and control logic clock must be stretched.

**CPUCLK (Output)**

This is a free running 25-MHz clock that is used for much of the sequential control logic. Although the name may imply it, this clock is not used by the CY7C611A CPU.

**CPUHCLK (Output)**

This is a 25-MHz stretched version of CPUCLK. It is the clock used to control the CY7C611A, CY7C602A, and address decode/latch logic. This clock is stretched by the CY7C361 if the address decoding logic reports that a slow, asynchronous cycle should be performed. When this clock is operating, it is always in phase with CPUCLK.

**CPU90 (Output)**

This is a 25-MHz free-running clock that lags the CPUCLK by 90 degrees (1/2 cycle). This clock, in conjunction with CPUCLK, provides the board control logic with a time base with 10 ns of resolution.

**START (Output)**

The assertion of START (Low) informs the slave 22V10B state machine that a 680x0 cycle should begin. This signal is not actually an output of the state machine, but of external state logic that is controlled by this state machine.

The 680x0 cycle cannot start at the beginning of the stretched clock cycle because of the latency associated with the assertion of INULL and FNULL from the CY7C611A and CY7C602A. If the NNULL signal is not asserted 40 ns after the clock stretching has started, the bus cycle is deemed valid and the START is asserted.

The state diagram of this machine is shown in *Figure 6*.

The transition equations for this machine are:

1. (State3) OR (HOLD AND /MHOLDA AND LWRT)
2. State3 AND /HOLD AND /MHOLDA AND /LWRT
3. State7 AND /DONE AND NNULL
4. (State7) OR (DONE AND /NNULL)

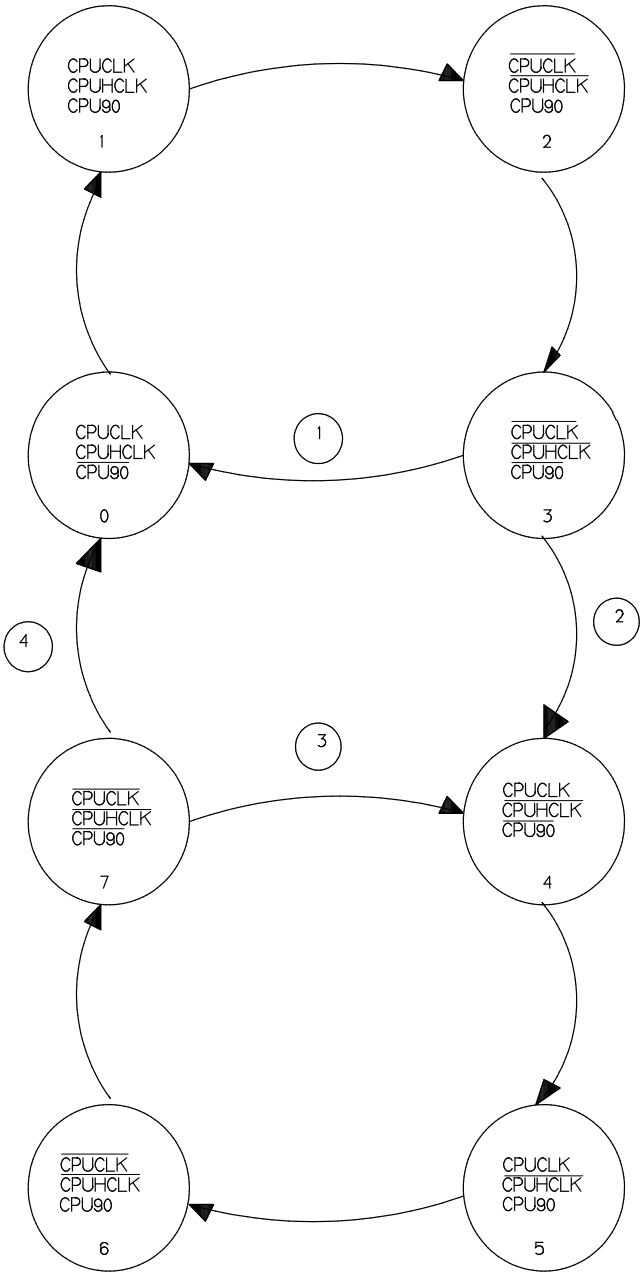



**Figure 6. Clock-Generation State Machine**

## Clock-Stretch Machine Operation

This machine has two main paths of operation. The first is a sequence in which all three clock outputs are operating, sequencing through states 0, 1, 2, 3, and back to 0. The second is a clock-stretched path sequencing through states 4, 5, 6, 7, and back to 4. This machine enters state 0 at the deassertion of reset and therefore always begins execution generating all clocks.

While within state 3, the machine samples the HOLD, $\overline{\text{MHOLDA}}$, and LWRT (Latch Advanced Write signal). HOLD High indicates that memory address on the bus is not selecting a slow device and that the clock should not be stretched. $\overline{\text{MHOLDA}}$ Low in this circuit indicates that a memory exception has occurred, and that the processor clock should continue to operate. The clock must be re-enabled so that $\overline{\text{MDS}}$ and $\overline{\text{MEXC}}$ can strobe the memory exception condition into the device.

The third signal sampled is Latched Advanced Write (LWRT). When this signal is High it indicates that the processor is starting a store cycle. The CY7C611A does not provide data to be stored until the second clock cycle of the operation. Therefore the processor must be advanced by at least one clock to place the data on the external bus. LWRT High and MHOLDA Low cause the processor clock to continue operating even if the address decoding logic asserts HOLD Low, indicating that the cycle should be stretched.

If HOLD is asserted (Low) and neither LWRT or MHOLDA are in their active states, then the state machine moves to state 4 and the clock is disabled to the processor and control logic. The other output clocks (CPUCLK and CPU90) continue to operate as the machine sequences through states 4, 5, 6, and 7. State 7 is also a decision-making state within this machine. At this point the machine either continues stretching or re-enables the processor and control logic clock. The clock is only re-enabled if either of two conditions (NNULL or DONE are detected active (Low)) is true.

NNULL is asserted if the current cycle is not a store cycle and the CY7C611A or CY7C602A nullifies the cycle. The processor does not generate the INULL or FNULL until late in the cycle. Therefore nullified asynchronous bus cycles end up being stretched for 40 ns before this is determined. If the stretched bus cycle is nullified by the CY7C611A or CY7C602A, the NNULL is asserted before the machines samples the signal in state 7. If NNULL has not been asserted upon entering state 7 for the first time after clock stretching has begun, START is as-

serted (Low) . This signals the slave 22V10B that a 680x0 style cycle should begin.

## 680x0 Bus Cycle Machine

A Mealy state machine implemented in a 22V10B performs the 680x0-compatible asynchronous bus cycle and asserts $\overline{\text{MHOLDA}}$, $\overline{\text{MDS}}$, and $\overline{\text{MEXC}}$ to the CY7C611A if a bus error is detected. This machine uses the CY7C361 to synchronize all asynchronous signals and therefore operates in a totally synchronous environment. This simplifies the implementation of the machine and enhances performance.

The input and output signals for the machine are:

**START (Input)**

This signal is asserted (Low) by the CY7C361 clock control state machine to indicate that a Motorola bus cycle should start. This signal remains asserted until the bus cycle completes.

**LRD (Input)**

This is the latched Read Access (RD) signal from the CY7C611A.

**SPARC_WB (Input)**

This is an output from the local bus arbiter for the board. If the asynchronous bus cycle is accessing something on the shared half bus, this signal is asserted by the address decode logic. If this signal is active (Low) the bus cycle must not begin until the grant signal, SPARC_GB, is asserted, granting access to the shared bus.

**SPARC_GB (Input)**

This active-Low signal is the bus grant signal from the local bus arbiter.

**DONE (Input)**

This is the synchronous active-Low signal from the CY7C361, indicating that some form of asynchronous cycle acknowledge has been received.

**BERR (Input)**

An asynchronous active-Low input that is combined with DONE for synchronization.

**AS (Output)**

This is the active-Low 680x0 compatible address strobe.

**DS (Output)**

This is the active-Low 680x0 compatible data strobe

**$\overline{\text{MHOLDA}}$ (Output)**

This is the $\overline{\text{MHOLDA}}$ signal to freeze the pipeline of the CY7C611A and CY7C602A.

**$\overline{\text{MDS}}$ (Output)**

This output is the $\overline{\text{MDS}}$ and $\overline{\text{MEXC}}$ signals to the CY7C611A. Since this bus control cycle only requires $\overline{\text{MDS}}$ for memory exception cycles, it was possible to reduce these two into a single output on the machine.

The state diagram of the machine is shown in *Figure 7*.

The transition equations for this machine are:

1. (/START AND /LRD AND /SPARC_WB AND /SPARC_GB) OR (/START AND /LRD AND SPARC_WB)
2. (/START AND LRD AND /SPARC_WB AND /SPARC_GB) OR (/START AND LRD AND SPARC_WB)
3. /DONE AND /BERR
4. /DONE AND BERR

## Machine Operation

This machine resets to state 0 and waits for the assertion of the START signal from the CY7C361. When this signal is active, the machine samples the states of the CY7C611A Latched Read Access signal (LRD), the Local Bus Request signal (SPARC_WB), and the Local Bus Grant signal (SPARC_GB). The state of LRD instructs the 22V10B to perform either a read or write cycle. If SPARC_WB is asserted (Low), then the

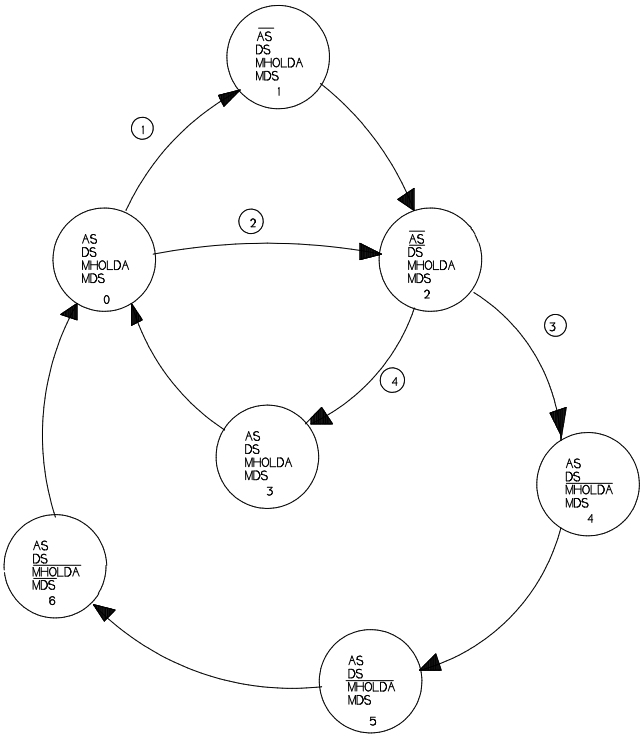



**Figure 7. 680x0 Bus Cycle State Machine**

peripheral device being accessed is on the shared section of the board. The bus cycle cannot start until a grant has been issued by the local bus arbiter. When SPARC_GB is asserted (Low), the shared bus has been granted to the CY7C611A. Read or write cycles that access peripherals on the local section of the card (CY7C611A access only) do not need permission from the local bus arbiter. CY7C611A local accesses can occur simultaneously with VIC64 local DMA accesses.

When the conditions have been met to start a cycle, AS strobe is asserted (Low). If the cycle is a read, DS is also asserted (Low). On write cycles, the assertion of DS is delayed one clock cycle to mimic the 680x0 cycle. This may not be necessary for many peripherals since, unlike Motorola processors, the CY7C611A has already placed the data on the data bus before the assertion of AS.

The machine moves to state 2 were it waits for the assertion of DONE (Low) from the CY7C361. DONE is generated by combining the board's asynchronous peripheral acknowledge and bus error signals. This combined signal is then run through the double input register structure of the CY7C361 to synchronize it. Double registering these asynchronous signals with the CY7C361 is the most efficient manner of synchronization as these registers are being clocked internally at 100 MHz. The entire double-register synchronization process takes only 20 ns. DONE is further qualified with the appropriate 25-MHz clock from the CY7C361 so that it can be considered completely synchronous to the 22V10B.

When the 22V10B detects DONE asserted, it samples the BERR input. If BERR is inactive (High), then the cycle terminates normally. The machine drives AS and DS inactive and advances back to through state 3 to state 0 to prepare for the next cycle. State 3, a delay state, is necessary to allow the control logic recovery time before the next cycle begins. This is a Mealy machine and removing state 3 would allow situations to occur were AS and DS would not meet the minimum High times required by slow peripherals. Refer to the waveforms on the following pages, which show the control signal sequencer for normally terminated and memory exception cycles.

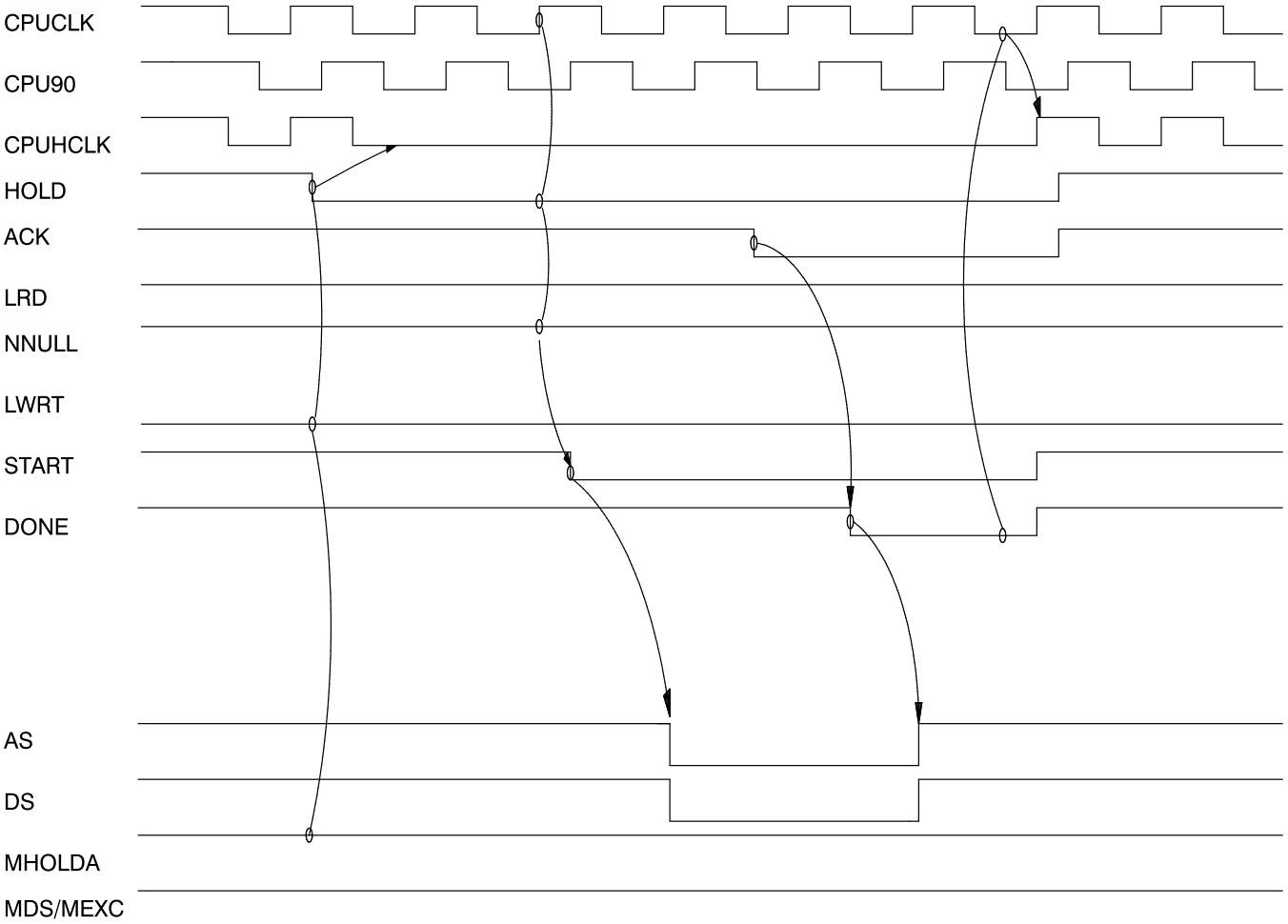
If BERR is active (Low) when DONE is asserted, the asynchronous cycle is terminated in a bus error or memory exception. The 22V10B asserts MHOLDA (Low) to the CY7C611A freezing the pipeline. The assertion of this signal informs the CY7C361 to re-enable clocking to the CY7C611A and control logic so that the exception can be strobed in using $\overline{\text{MDS}}$ and $\overline{\text{MEXC}}$. $\overline{\text{MDS}}$ and $\overline{\text{MEXC}}$ are then asserted simultaneously to the CY7C611A, indicating that a memory exception has occurred. Since memory exceptions are the only cycles that freeze the CY7C611A pipeline, $\overline{\text{MDS}}$ and $\overline{\text{MEXC}}$ are always asserted simultaneously. This allows the generation of a single signal, rather than two, freeing up an output on the PLD.

## Conclusion

This hybrid bus conversion has worked well on the CY7C611A/VIC64 VMEbus board. Using the CY7C361 as a clock-generation device, allowing all of the logic on the board to operate from the relatively slow 25-MHz clocks, greatly simplified the timing analysis without sacrificing performance. The CY7C361 also provides logic functions that are not discussed within this application note. In many applications it may be possible to move the slave 680x0 bus-cycle generation state logic into the CY7C361. This would reduce the bus conversion logic to a single device.

## Output Waveforms

**CY7C611A to 680x0 Normally Terminated Cycle**



## Output Waveforms (continued)

**CY7C611A to 680x0 Memory Exception Cycle**