

CSCE 491

Computer Engr. Design Project

2005/9/6

Week 5

Modeling & Design Applications-2

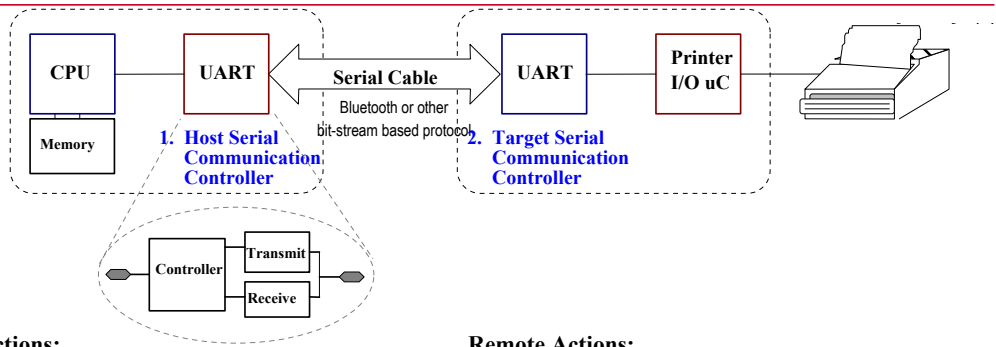
The UART

© 2002 Dr. James P. Davis

UART Function Definition-1

- Purpose.
 - ✓ Handle half-duplex serial data traffic between two connection points of a communications channel. Most UARTs are full-duplex, handling both send and received traffic simultaneously. We limit the scope to minimize complexity, to create a design model on which we can layer additional capabilities later.
 - ✓ This gives a very basic model of transmit/receive protocol handshaking, that will be the basis for examining the more complex handshaking associated with the 802.11 protocol.
- Algorithmic approach.
 - ✓ Use a pipelined “delegation” scheme, where the CPU enables the UART controller, and the controller enables either transmitter or receiver.
 - ✓ Data transfer between sender and receiver is done according to a 4-stage handshaking protocol:
 - ✓ RTS: sender endpoint requests to send data
 - ✓ CTS: receiver indicates that sender is “clear” to send the data stream.
 - ✓ The actual data is sent as a bit serial data stream, and the data carries additional parity bits.
 - ✓ ACK: receiver acknowledges that the data was received successfully.
 - ✓ We adopt a sequence of polling loops to synchronize activities of the protocol between sender and receiver end points.
- ASM diagram.
 - ✓ We create a design with 3 threads, plus a 4th thread that provides an abstraction of the CPU interface (with limited functionality to initiate activity in the UART blocks)
 - ✓ We'll use this 4th thread as a test harness.

UART Function Definition-2



Host Actions:

1. Set master enable; set control flags; load 32-bit data word for transmit sequence (LSB is parity bit).
3. Load 'transmit data' register; buffer with start, stop and parity bits; set 'ready to send' flag.
5. Reset counter; shift each of 10-bits onto serial bus; increment transfer counter.
8. Check and clear status flags; setup for next transmit.

Remote Actions:

2. Clear flags on Reset; drop 'clear to send' flag; poll for 'ready to send'.
4. Read 'ready to send' and acknowledge with 'clear to send' flag; set up the 'receive data counter'.
6. Shift in each of 10-bits from serial bus into buffer; check parity; strip off start and stop bits.
7. Check and clear status flags; transfer data to buffer.

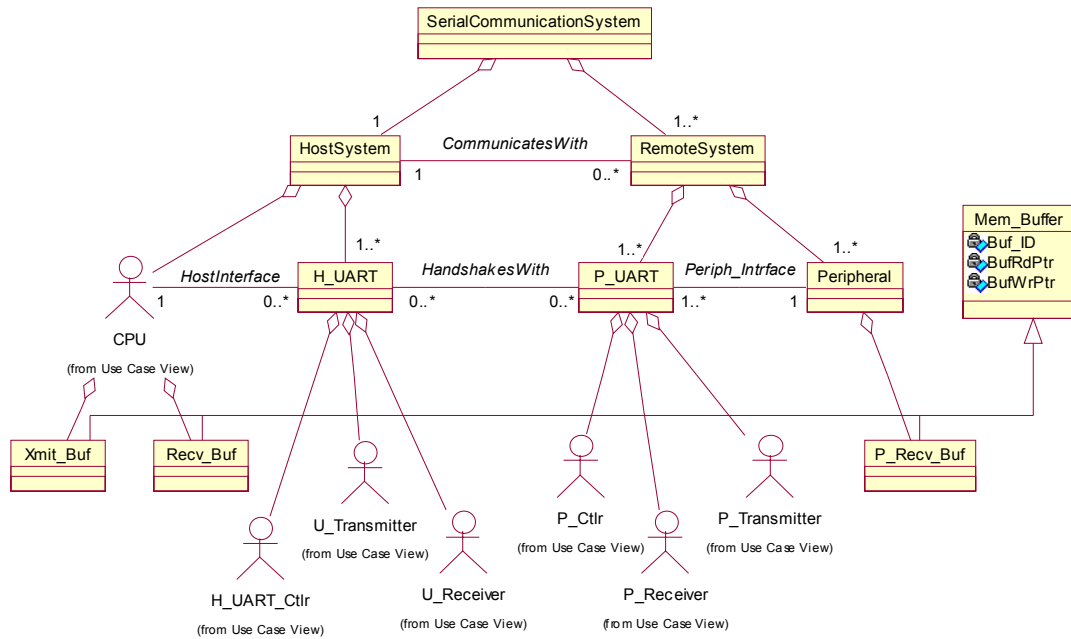


UART – Functional Description

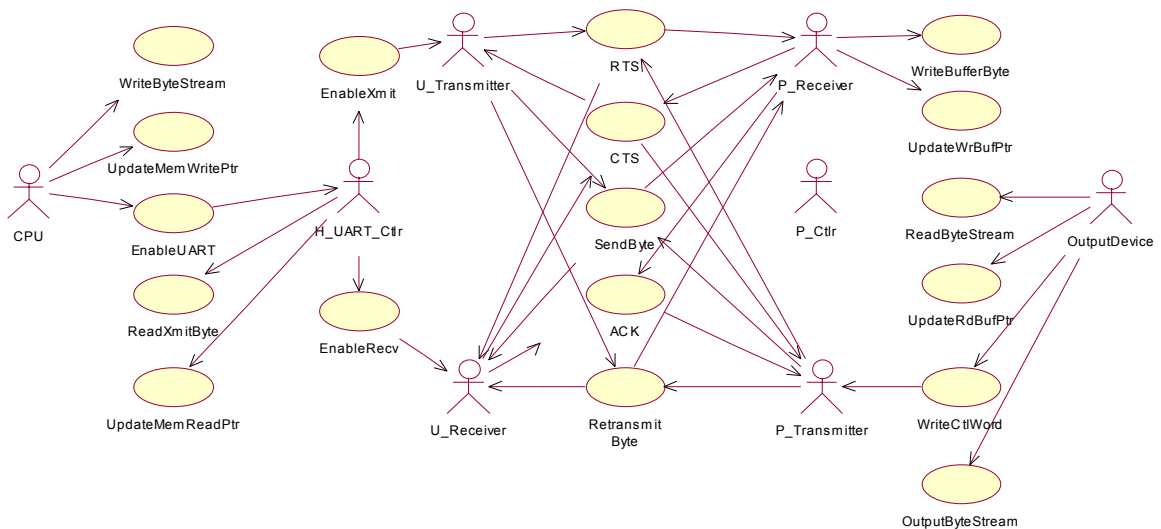
1. A universal Asynchronous Receiver Transmitter (UART) provides asynchronous data link between a microprocessor and external peripheral not connected directly to its internal buses. UART contains receiver, transmitter and centralized control unit modules. This UART is derived from the Motorola MC6850 ACIA.
2. Control Unit processes commands and moves data to/from the CPU, as well as other functions, e.g., clock division, interrupts, and setting stop bits, parity type (even, odd) based on the control register bit settings. Currently, we assume 1 stop bit.
3. Receiver reads serial bit stream input from remote end of serial connection and converts data into a parallel word using a shift register. Afterwards, the start, stop, and parity bits are stripped before data word placed in Receiver Data Register (RDR). **HW #5:** Add the logic to the receiver so that it also strips the start, stop, and parity bits from this data frame prior to passing the data word; also, perform the parity check prior to sending the word.
4. Transmitter converts parallel data from Transmit Data Register (TDR) to serial form by shifting one bit per cycle, then transmits one bit at a time on serial output line, TxData1. **HW #5:** Add the logic to the transmitter so that it also adds the necessary start, stop, and parity bits to this data frame prior to transmission.



UART Class Diagram – Relevant Concepts



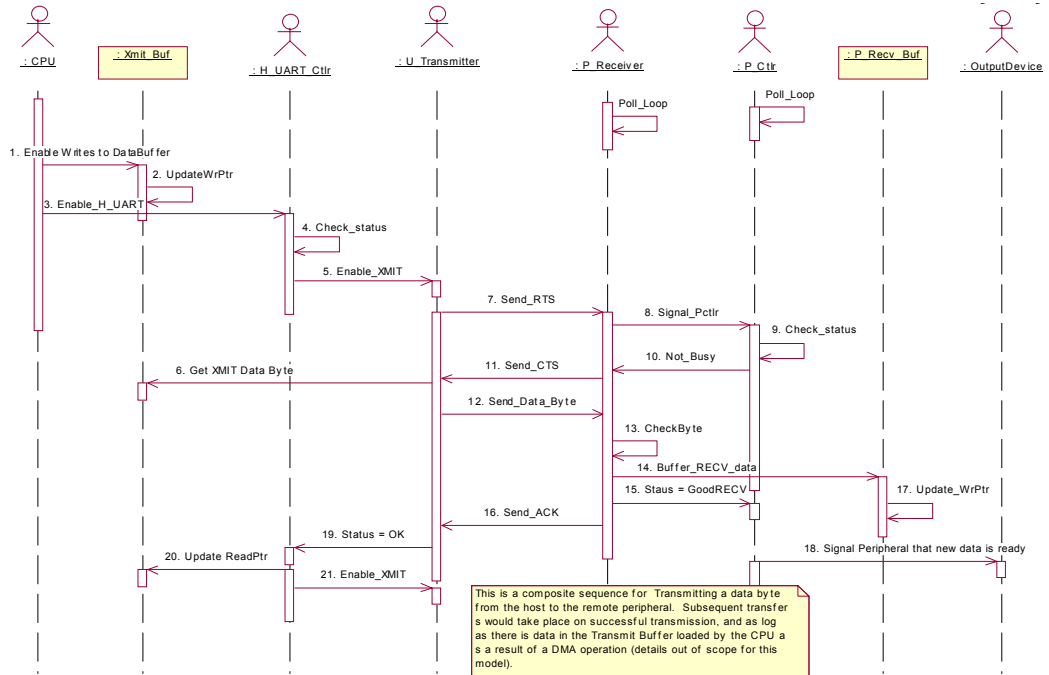
UART Use Case Diagram - Function Inventory



- Actors may represent individual function blocks, or roles played by modules in the system.
- Use Cases represent functions or transactions that are carried out between the Actors in the system.
- The architectural analysis activity here is to decompose and refine (through iteration) our understanding of the system until we are ready to commit to a detailed hardware function partitioning and design.

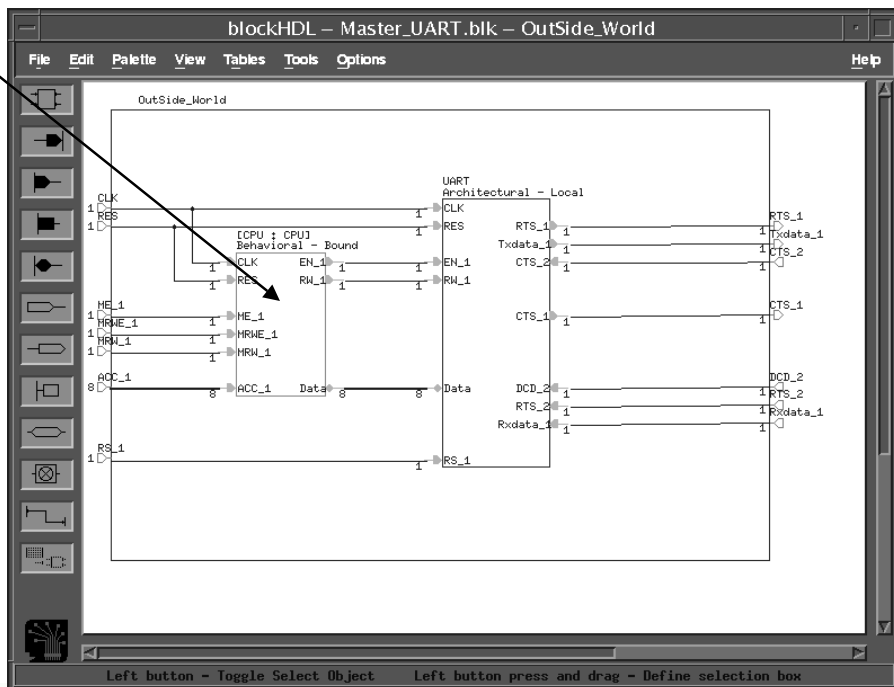


UART Sequence Diagram – Detailed Interactions

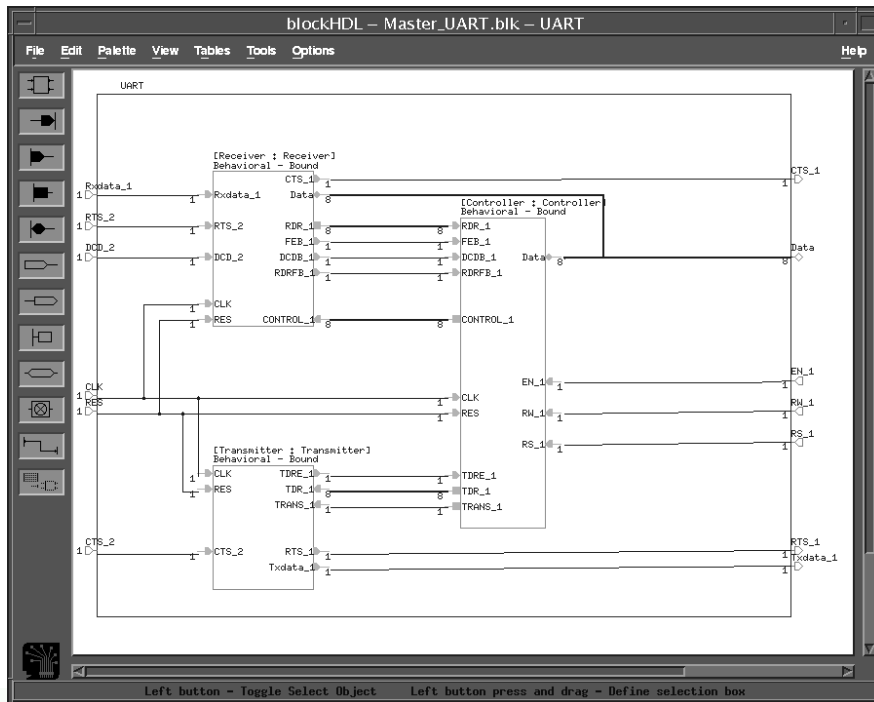


UART Architecture – Top Level Blocks

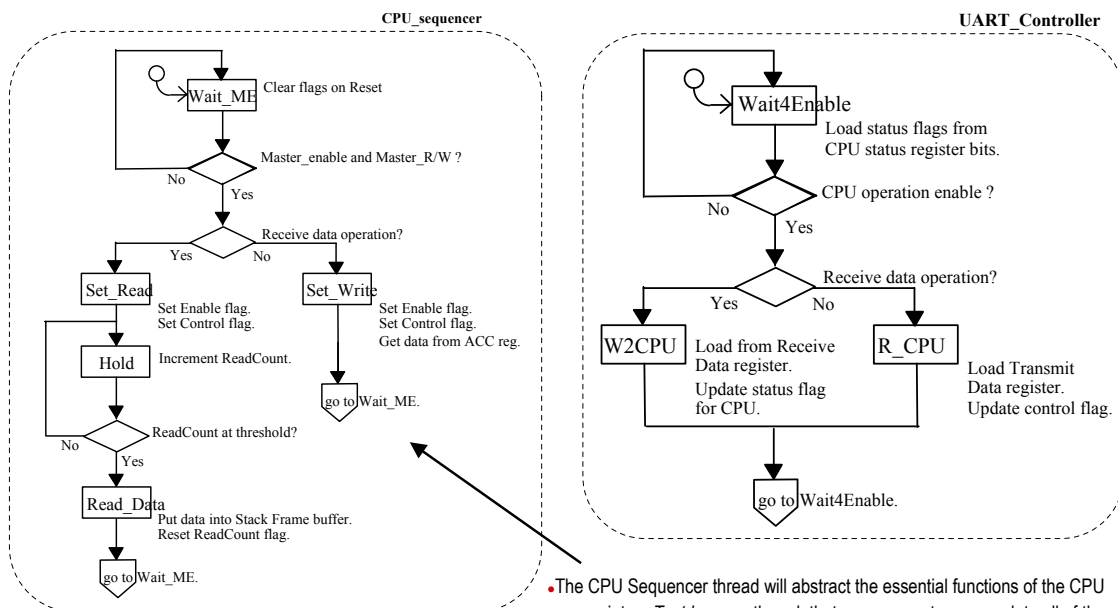
- We'll define a test harness to drive the hardware design "model under test".



UART Architecture – Blocks and Interfaces



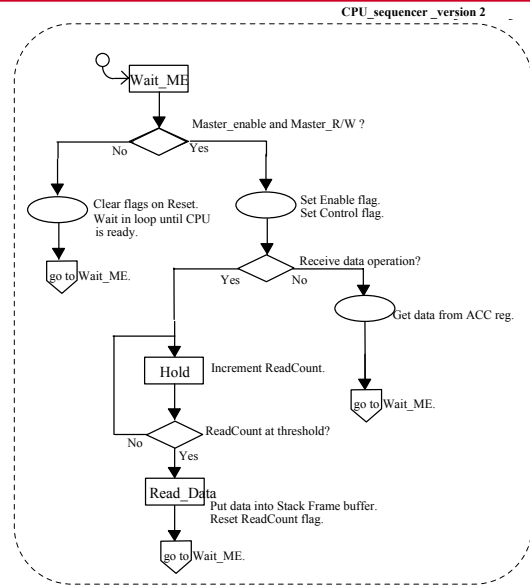
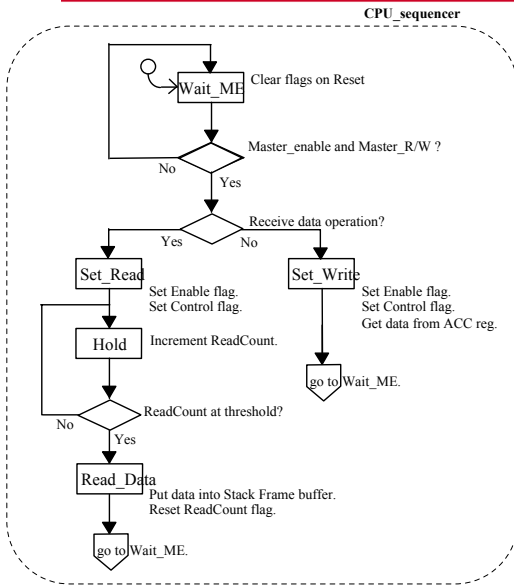
UART Architecture – Flowchart Pseudocode



•The CPU Sequencer thread will abstract the essential functions of the CPU into a *Test harness* thread, that we can use to encapsulate all of the control signals we need to instantiate to test values during simulation.



UART Architecture – Flowchart Pseudocode

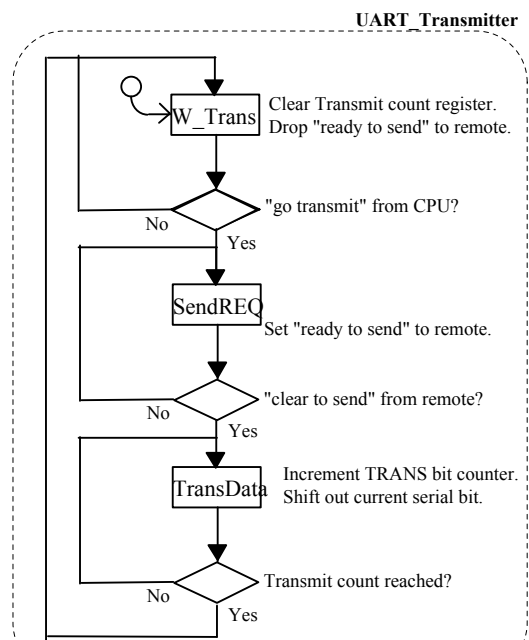
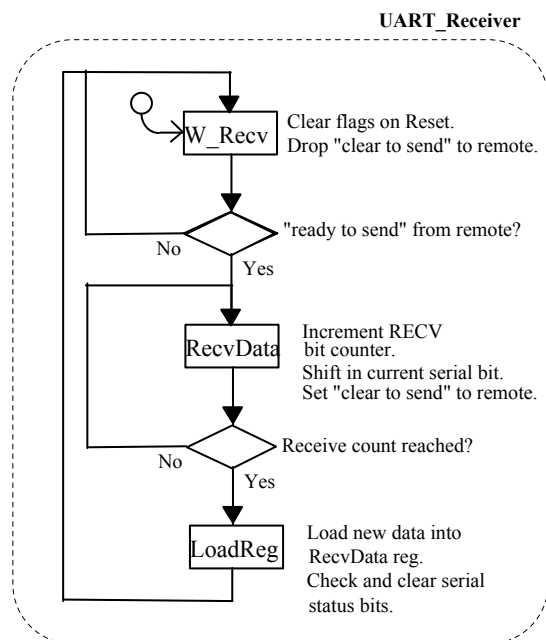


In this version, we make design more efficient by easily changing the "flow" of logic. With Mealy-style outputs, we eliminate 2 states.

=> Save a state register and save one clock cycle!



UART Architecture – Flowchart Pseudocode

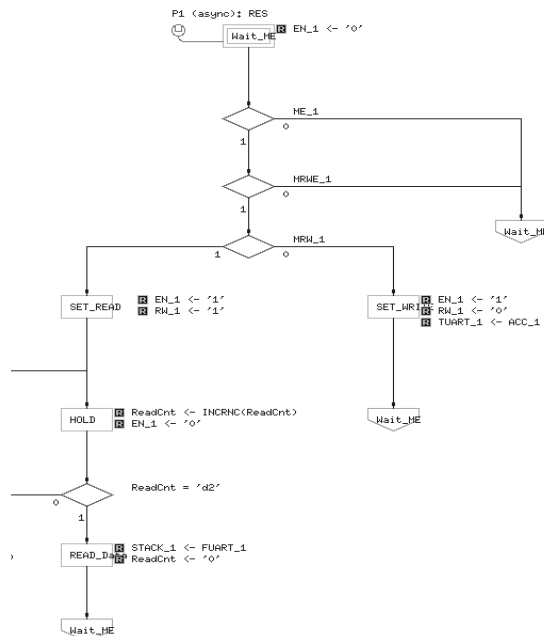


UART Signal/Bus Definitions

Bus Name	Type	Size	Description
ME	INPUT	1	Used to enable CPU
MRWE	INPUT	1	Used to enable CPU read/write
MRW	INPUT	1	Used to indicate a read or write cycle
ACC	INPUT	8	Used to input data to system
RS	INPUT	1	Used to select UART registers
RxData	INPUT	1	Used to enter serial data to the systems
RTS_2	INPUT	1	Used to enable the Receiver
CTS_2	INPUT	1	Used to inform transmitter to go with transmission
DCD_2	INPUT	1	Lets receiver know that carrier is present
EN	INTERNAL	1	Used by CPU to activate the UART
RW	INTERNAL	1	Used by store data from the CPU to transmitted
TDR	INTERNAL	8	Used to store data from the CPU to transmitted
RDR	INTERNAL	8	Used to store deserialized data from the Receiver
TShift	INTERNAL	10	Used by Transmitter to serialize data from the TDR
Rshift	INTERNAL	10	Used by Receiver to store incoming serial data
STATUS	INTERNAL	8	Used to store the status information in the UART
CONTROL	INTERNAL	8	Used to store the UART systems parameters
RCNT	INTERNAL	4	Receiver count loop control signal
TCNT	INTERNAL	4	Transmitter count loop control signal
Txdata	OUTPUT	1	Transmitter's serial output
TRANS	INTERNAL	1	Used by Controller to enable the Transmitter
STACK	INTERNAL	8	Houses data from UART in the CPU
RTS_1	OUTPUT	1	Used by Transmitter to activate outside receiver
DCD_1	OUTPUT	1	Used by UART to indicate a carrier
CTS_1	INTERNAL	1	Used by Receiver to indicate transfer is OK
TUART	INTERNAL	8	Used by CPU to send data to UART
FUART	INTERNAL	8	Used by UART to send data back to CPU
RDRFB	INTERNAL	1	Used by Receiver to indicate new data in the RDR
TDRF	INTERNAL	1	Used by Transmitter to indicate the TDR is empty
FEB	INTERNAL	1	Used by Receiver to indicate improper protocol
DCDB	INTERNAL	1	Used by Receiver to indicate a carrier present



ASM Diagram – CPU Interface



● Purpose.

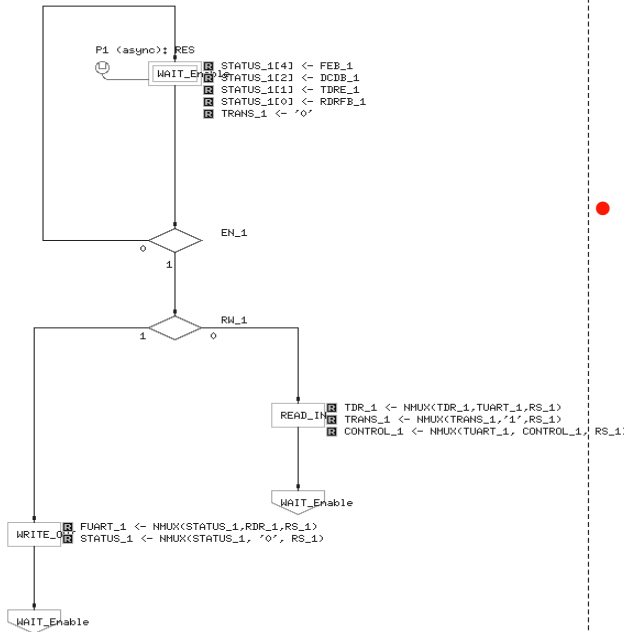
- ✓ We choose to create an abstract model of the CPU interface. Since our focus is on the UART itself, and not the CPU, we need only model that behavior of the CPU relevant to execution of the UART.

● CPU Interface.

- ✓ The CPU enables the UART controller.
- ✓ EN is enable signal to start the execution of the CPU thread.
- ✓ ME: Master Enable – to kick off the UART controller.
- ✓ MRWE: Master Read/Write enable – to enable the UART controller for the Read/Write functions (there are other UART function modes, but we are not modeling these here).
- ✓ MRW: Master Read/Write select flag - to tell the UART whether the operation will be a Read or Write.



UART Architecture - Controller



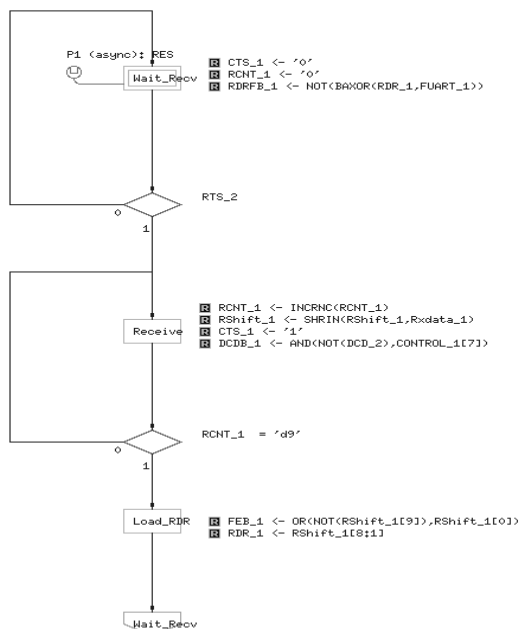
Purpose.

- ✓ We model the basic control functions of the UART first. Since most of its capabilities deal with transmit and receive of data, we model these basic functions.

UART Controller.

- ✓ The Controller waits to be awakened and selected by the CPU. It loops in the WAIT_Enable state until the EN_1 signal goes high.
- ✓ The RW_1 signal sets whether the Controller will initiate a read or write operation. This is handled after testing the RW_1 signal, and setting control signals for the Transmit and Receive threads of the UART.

UART Architecture - Receiver



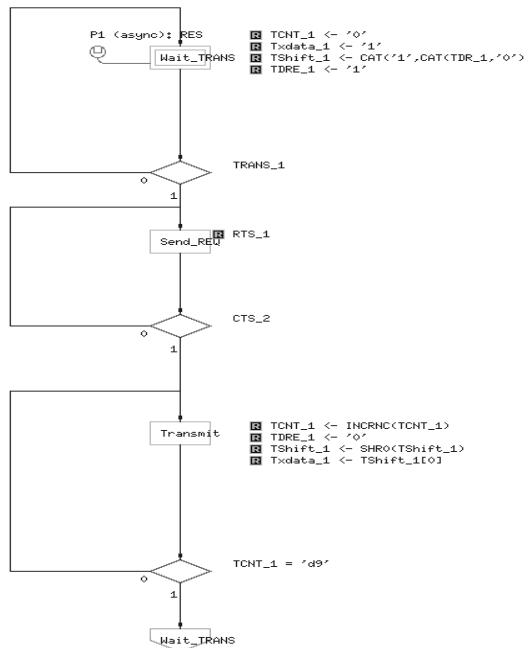
Purpose.

- ✓ We model the basic Receiver functions, most of which deal with waiting for a new data stream and reading this stream, shifting the data, then signaling the Controller.

UART Receiver.

- ✓ We start up in a poll loop, waiting for the RTS signal, indicating we have a request from the remote host to send data to us.
- ✓ The RTS_2 signal controls whether we initiate preparation to receive data. One thing we do is set the CTS_1 flag, initialize the RCNT counter, and shift in the serial bit.
- ✓ We'll loop on the reading and shifting until the RCNT flag indicates we have a full data word (1 start bit, 7 ASCII bits, and 1 parity bit).
- ✓ We then load the Read Data register RDR, and check the start and parity bits.

UART Architecture - Transmitter



- Purpose.
 - ✓ We model the basic Transmitter functions to send data to the remote unit across the serial line.
- UART Transmitter.
 - ✓ The Transmitter waits to be signaled by the Controller, by the TRANS_1 signal. A number of signals are sampled in Wait_TRANS state continuously.
 - ✓ When enabled, the RTS for request to send signal is set, and we go into a poll loop awaiting the remote side to pull the CTS_2 line, where we will let go of the RTS_1 line.
 - ✓ We go into the Transmit poll loop, cycling this state to send out data, bit by bit, incrementing the bit counter on each cycle.
 - ✓ We complete when the counter reaches word length.

© 2002 Dr. James P. Davis Page 17

UART - Design Verification

CPU: Simulation begins with CPU setting various control bits in the UART control register, illustrating the method in which data is transferred to the UART from the world and the ability for one component to control another through control signals.

Controller: After the master user places the data input on ACC bus and engages the CPU, the CPU enables the UART and sets the control lines, RS and RW, thus providing the UART instructions for properly dealing with the input.

Transmit: Once the control register is initialized, a transmit operation is executed. Again, the master user enables the CPU and provides data on the ACC bus. The UART Controller is engaged and the RS and RW lines are set to invoke a transmit operation. The Controller transfers the data from the TUART bus to the Transmit Data Register, TDR, and the pulses TRANS, which enables the Transmitter. The Transmitter places the contents of the TDR to its shift register and begins iterative shifts on TShift, placing the LSB of the shift register on the TXdata line.

Receive: The master user sets the RTS_2 input high indicating that remote device wishes to send data, and the Receiver is enabled. The Rxddata line is sampled ten times, one start bit, eight data bits, and one stop bit, and each bit is placed in its shift register. After the tenth sample, the RDRFB is set high to indicate the RDR has new data, and the data is transferred to the RDR after stripping off the start and stop bits. The CPU then engages a read cycle to retrieve the new data from the RDR, thus clearing the RDRFB signal.

Parallel Transmit and Receive: To prove that each of the components of the system may act in parallel, a simultaneous transmit and receive situation is initiated by the master user. The CPU is given the proper instructions to engage the UART Transmitter, at the same time, the RTS2 input goes high indicating a request to send data. A read cycle is performed after the receive operation to retrieve the new data from the RDR.



© 2002 Dr. James P. Davis Page 18

UART – Summary

- The UART is a familiar component we have seen before in CPU-based peripheral interfacing applications.
 - ✓ CSCE 212 (Computer Architecture), CSCE 313 (Embedded Systems)
- We start with a basic abstract model, and develop a set of analysis “artifacts” for this component, including a test harness we’ll use to drive the design through a set of test scenarios.
 - ✓ The scenarios will come from analysis we do using Use Case and Sequence Diagrams.
 - ✓ We’ll build support for these test cases into an ASM thread for the CPU (a very abstract model of the CPU function as seen from its interface).
- Over the next few assignments, we’ll develop our understanding of this model, by refining and elaborating with more detailed functionality.
 - ✓ This serial bitstream communication model forms the basic pattern for our architecture analysis of the 802.11b protocol engine.

