Left: E:/Programas/Arduino1.8.8/hardware/teensy/avr/cores/teensy3/serial2.c.original
Right: E:/Programas/Arduino1.8.8/hardware/teensy/avr/cores/teensy3/serial2.c

Left: Wed Dec 26 06:32:05 2018	Right: Thu Sep 12 19:31:21 2019
Line 61	**Line 61**
static volatile BUFTYPE tx_buffer[SERIAL2_TX_BUFFER_SIZE];	static volatile BUFTYPE tx_buffer[SERIAL2_TX_BUFFER_SIZE];
static volatile BUFTYPE rx_buffer[SERIAL2_RX_BUFFER_SIZE];	static volatile BUFTYPE rx_buffer[SERIAL2_RX_BUFFER_SIZE];
static volatile uint8_t transmitting = 0;	static volatile uint8_t transmitting = 0;
	static volatile uint8_t queueing_tx_data = 0; // PedroR
#if defined(KINETISK)	#if defined(KINETISK)
static volatile uint8_t *transmit_pin=NULL;	static volatile uint8_t *transmit_pin=NULL;
#define transmit_assert() *transmit_pin = 1	#define transmit_assert() *transmit_pin = 1
Line 117	**Line 118**
tx_buffer_head = 0;	tx_buffer_head = 0;
tx_buffer_tail = 0;	tx_buffer_tail = 0;
transmitting = 0;	transmitting = 0;
	queueing_tx_data = 0; // PedroR
#if defined(KINETISK)	#if defined(KINETISK)
switch (rx_pin_num) {	switch (rx_pin_num) {
case 9: CORE_PIN9_CONFIG = PORT_PCR_PE | PORT_PCR_PS | PORT_PCR_PFE | PORT_PCR_MUX(3); break;	case 9: CORE_PIN9_CONFIG = PORT_PCR_PE | PORT_PCR_PS | PORT_PCR_PFE | PORT_PCR_MUX(3); break;
Line 352	**Line 355**
uint32_t head, n;	uint32_t head, n;
if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;	if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;
	// PedroR: protect this code block
	queueing_tx_data = 1;
if (transmit_pin) transmit_assert();	if (transmit_pin) transmit_assert();
head = tx_buffer_head;	head = tx_buffer_head;
if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;	if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;
Line 373	**Line 380**
tx_buffer[head] = c;	tx_buffer[head] = c;
transmitting = 1;	transmitting = 1;
tx_buffer_head = head;	tx_buffer_head = head;
	queueing_tx_data = 0;
	// PedroR: end protecting code block
	// An interrupt can still occur between clearing the flag and setting UART_C2; therefore we will also check for
	// (!TX_BUFFER_EMPTY) in the TC interrupt, in addition to the queueing_tx_data flag
UART1_C2 = C2_TX_ACTIVE;	UART1_C2 = C2_TX_ACTIVE;
}	}
Line 384	**Line 397**
uint32_t head, n;	uint32_t head, n;
if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;	if (!(SIM_SCGC4 & SIM_SCGC4_UART1)) return;
	// PedroR: protect this code block
	queueing_tx_data = 1;
if (transmit_pin) transmit_assert();	if (transmit_pin) transmit_assert();
while (p < end) {	while (p < end) {
head = tx_buffer_head;	head = tx_buffer_head;
if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;	if (++head >= SERIAL2_TX_BUFFER_SIZE) head = 0;
if (tx_buffer_tail == head) {	if (tx_buffer_tail == head) {
	queueing_tx_data = 0; // Pedro R: need to be 0 so that C2_TX_ACTIVE fires the necessary interrupts properly
UART1_C2 = C2_TX_ACTIVE;	UART1_C2 = C2_TX_ACTIVE;
do {	do {
int priority = nvic_execution_priority();	int priority = nvic_execution_priority();
Line 405	**Line 423**
yield();	yield();
}	}
} while (tx_buffer_tail == head);	} while (tx_buffer_tail == head);
	queueing_tx_data = 1; // Pedro R: restore protection to code block
}	}
tx_buffer[head] = *p++;	tx_buffer[head] = *p++;
transmitting = 1;	transmitting = 1;
tx_buffer_head = head;	tx_buffer_head = head;
}	}
	queueing_tx_data = 0;
	// PedroR: end protecting code block
	// An interrupt can still occur between clearing the flag and setting UART_C2; therefore we will also check for
	// (!TX_BUFFER_EMPTY) in the TC interrupt, in addition to the queueing_tx_data flag
UART1_C2 = C2_TX_ACTIVE;	UART1_C2 = C2_TX_ACTIVE;
}	}
#else	#else
Line 553	**Line 580**
}	}
}	}
c = UART1_C2;	c = UART1_C2;
if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) {	if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) { // uart tx FIFO below watermark (as if empty)
head = tx_buffer_head;	head = tx_buffer_head;
tail = tx_buffer_tail;	tail = tx_buffer_tail;
do {	do {
Line 562	**Line 589**
avail = UART1_S1;	avail = UART1_S1;
n = tx_buffer[tail];	n = tx_buffer[tail];
if (use9Bits) UART1_C3 = (UART1_C3 & ~0x40) | ((n & 0x100) >> 2);	if (use9Bits) UART1_C3 = (UART1_C3 & ~0x40) | ((n & 0x100) >> 2);
UART1_D = n;	UART1_D = n;
} while (UART1_TCFIFO < 8);	} while (UART1_TCFIFO < 8);
tx_buffer_tail = tail;	tx_buffer_tail = tail;
if (UART1_S1 & UART_S1_TDRE) UART1_C2 = C2_TX_COMPLETING;	if (UART1_S1 & UART_S1_TDRE) {
	// PedroR
	if (queueing_tx_data) {
	UART1_C2 = C2_TX_INACTIVE; // the code block protected by queueing_tx_data will address UART_C2 and re activate it
	} else {
	UART1_C2 = C2_TX_COMPLETING;
	}
	// End Pedro R
	}
}	}
#else	#else
if (UART1_S1 & UART_S1_RDRF) {	if (UART1_S1 & UART_S1_RDRF) {
Line 585	**Line 620**
if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) {	if ((c & UART_C2_TIE) && (UART1_S1 & UART_S1_TDRE)) {
head = tx_buffer_head;	head = tx_buffer_head;
tail = tx_buffer_tail;	tail = tx_buffer_tail;
if (head == tail) {	if (head == tail) { // PedroR: head == tail is the same as my TX_BUFFER_EMPTY
UART1_C2 = C2_TX_COMPLETING;	// PedroR
	if (queueing_tx_data) {
	UART1_C2 = C2_TX_INACTIVE; // the code block protected by queueing_tx_data will address UART_C2
	} else {
	UART1_C2 = C2_TX_COMPLETING;
	}
	// End Pedro R
} else {	} else {
if (++tail >= SERIAL2_TX_BUFFER_SIZE) tail = 0;	if (++tail >= SERIAL2_TX_BUFFER_SIZE) tail = 0;
n = tx_buffer[tail];	n = tx_buffer[tail];
Line 597	**Line 638**
}	}
#endif	#endif
if ((c & UART_C2_TCIE) && (UART1_S1 & UART_S1_TC)) {	if ((c & UART_C2_TCIE) && (UART1_S1 & UART_S1_TC)) {
transmitting = 0;	
if (transmit_pin) transmit_deassert();	
UART1_C2 = C2_TX_INACTIVE;	
	// PedroR
	if (queueing_tx_data) {
	UART1_C2 = C2_TX_INACTIVE; // the code block protected by queueing_tx_data will address UART_C2
	} else if (tx_buffer_head != tx_buffer_tail) { // TX buffer not empty
	UART1_C2 = C2_TX_ACTIVE; // force interrupts to continue firing
	} else {
	transmitting = 0;
	if (transmit_pin) transmit_deassert();
	UART1_C2 = C2_TX_INACTIVE;
	}
	// End PedroR
}	}
}	}
}	}