G1_Infrared_data_to_Software

Jesse Larson, Jing Lu, and Qingqing Liu

The goal of this app note:

Give you a simple and easy way to get infrared commands to your software

How to set things up:

Essentially I used the PIO register. Firstly set the width to the amount of bits you want to send to software. Check the synchronously capture box, and set edge type to rising. Leave the enable bit-clearing for edge capture register unchecked. Check the generate IRQ box, and set the IRQ Type to edge. Add this altera_avalon_pio to your qsis system. Export your conduit end and attach these to the conduit export of your receiver module.

How your receiver needs to talk to the PIO:

Put simply once your receiver has detected a valid 8 bit data sequence, or whatever packet size you decide on, (1 to 32) is the valid range for a PIO register. It should drive a single pulse of this data on the PIO lines, then return to driving all zero, and start recording the next sequence of infrared bits.

What the PIO does is captures each of the rising edges and each data line and sets these bits in the edge capture register and sends an interrupt to the cpu (be sure to assign an interrupt vector to your pio). Put simply this lets you not have to worry about timing issues. If your software cannot read and clear the data in the edge capture register during the 100000+ clock cycles that pass before another potential valid 8 bit command could be driven to the PIO, your software has significant problems.

Setting up the software:

This code is directly taken from the count_binary.c file given by altera, the only additions made by me is the queue message setup. This code will set up your PIO register to raise an interrupt properly. However check out the original file to gain more understanding of the volatile edge capture pointer works to gain more intuition on how it functions.

```
/* messages for ISR routine */
#define N_RESOURCES             4
OS_EVENT *QSem;
void *QMsg[N_RESOURCES];

// stores edge capture of IR
volatile int edge_capture;
#ifdef IR_RUPT_INTERFACE_BASE

#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
static void handle_IR_interrupts(void* context)
#else
static void handle_IR_interrupts(void* context, alt_u32 id)
#endif
{
    /* Cast context to edge_capture's type. It is important that this be
     * declared volatile to avoid unwanted compiler optimization.
     */
    volatile int* edge_capture_ptr = (volatile int*) context;
    /* Store the value in the IR edge capture register in *context. */
    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(IR_RUPT_INTERFACE_BASE);
    /* Reset the IR edge capture register. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(IR_RUPT_INTERFACE_BASE, 0);
    // transfer register data to task
    alt_u32 content = * edge_capture_ptr;
    OSQPost(QSem, (void*) content);
    // small delay before exiting the interrupt
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(IR_RUPT_INTERFACE_BASE);
}
static void init_IR_pio()
{
    /* Recast the edge_capture pointer to match the alt_irq_register() function
     * prototype. */
    void* edge_capture_ptr = (void*) &edge_capture;
    /* Enable all 8 bit interrupts for edge capture register. */
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(IR_RUPT_INTERFACE_BASE, 0xff);
    /* Reset the edge capture register. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(IR_RUPT_INTERFACE_BASE, 0x0);
    /* Register the interrupt handler. */
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
    alt_ic_isr_register(IR_RUPT_INTERFACE_IRQ_INTERRUPT_CONTROLLER_ID,
IR_RUPT_INTERFACE_IRQ,
      handle_IR_interrupts, edge_capture_ptr, 0x0);
#else
    alt_irq_register( IR_RUPT_INTERFACE_IRQ, edge_capture_ptr,
      handle_IR_interrupts);
#endif
}
#endif
```

The following code should appear in your main

```
#ifdef IR_RUPT_INTERFACE_BASE
    init_IR_pio();
#endif
```

```
/* create OS message queue */
QSem = OSQCreate(&QMsg[0], N_RESOURCES);
```

The following is how you get your command in one of your tasks

```
INT8U err = OS_NO_ERR;
alt_u32 IR_DATA = (alt_u32) OSQPend(QSem, 0, &err);
```

Congratulations:

You have now successfully set up your software to receive data from your slow infrared receiver component.

Cheers. Jesse Larson, Jing Lu, and Qingqing Liu.