start_application()

start_application()主要完成Tcp Server的创建，包括绑定端口，开启Server监听，使用回调函数设置端口被Client连接之后的反馈。

在分析之前先声明一个概念，`protocol control block - PCB,` pcb是lwip内部特别定义的结构体，用于对网络传输协议(UDP、TCP)进行控制。

函数名称	函数作用
tcp_bind (struct tcp_pcb *pcb, ip_addr_t *ipaddr, u16_t port);	将Tcp控制模块pcb，与特定端口port连接
tcp_arg (struct tcp_pcb *pcb, void *arg);	设置传递给回调函数的参数
tcp_listen(pcb);	tcp_listen_with_backlog()的宏定义，开启监听模式
tcp_accept(pcb, accept_callback);	设置Server端口被连接之后触发的回调函数
accept_callback();	Server被Client连接后，触发该函数

```
int start_application()
{
  struct tcp_pcb *pcb;
  err_t err;
  unsigned port = 7;

  /* create new TCP PCB structure */
  pcb = tcp_new();
  if (!pcb) {
    xil_printf("Error creating PCB. Out of Memory\n\r");
    return -1;
  }

  /* bind to specified @port */
  err = tcp_bind(pcb, IP_ADDR_ANY, port);
  if (err != ERR_OK) {
    xil_printf("Unable to bind to port %d: err = %d\n\r", port, err);
    return -2;
  }

  //设置传递给回调函数的参数
```

```
  /* we do not need any arguments to callback functions */
  tcp_arg(pcb, NULL);

  //开启pcb的监听模式，server会一直等待client发起的连接请求
  /* listen for connections */
  pcb = tcp_listen(pcb);
  if (!pcb) {
    xil_printf("Out of memory while tcp_listen\n\r");
    return -3;
  }

  /* specify callback to use for incoming connections */
  tcp_accept(pcb, accept_callback);
  return 0;
}
```

进入回调函数**accept_callback()**之后，会直接调用一个新的回调函数，**recv_callback()**，用来对接收到的数据进行进一步处理，这里给出**recv_callback()**函数的简单分析。

函数定义如下，如果Tcp连接没有成功建立，就关闭当前的pcb。

如果建立了连接，并且收到数据，首先调用**tcp_recved()**函数增大当前Tcp控制模块的数据窗口。然后比较当前tcp的发送buffer的空间和接收数据的长度(接收数据存储在pbuf结构体中)。

如果**(tcp_sndbuf(tpcb) > p->len)**为真，就使用tcp_write()函数将接收到的数据**(p->payload)**写入到发送队列里，等待tcp的发送。

这样就实现了数据的Echo Back。

```
err_t recv_callback(void *arg, struct tcp_pcb *tpcb,
                               struct pbuf *p, err_t err)
{
  /* do not read the packet if we are not in ESTABLISHED state */
  if (!p) {
    tcp_close(tpcb);
    tcp_recv(tpcb, NULL);
    return ERR_OK;
  }

  /* indicate that the packet has been received */
  tcp_recved(tpcb, p->len);

  /* echo back the payload */
  /* in this case, we assume that the payload is < TCP_SND_BUF */
  if (tcp_sndbuf(tpcb) > p->len) {
    err = tcp_write(tpcb, p->payload, p->len, 1);
```

```
  } else
    xil_printf("no space in tcp_sndbuf\n\r");

  /* free the received pbuf */
  pbuf_free(p);
  return ERR_OK;
}
```